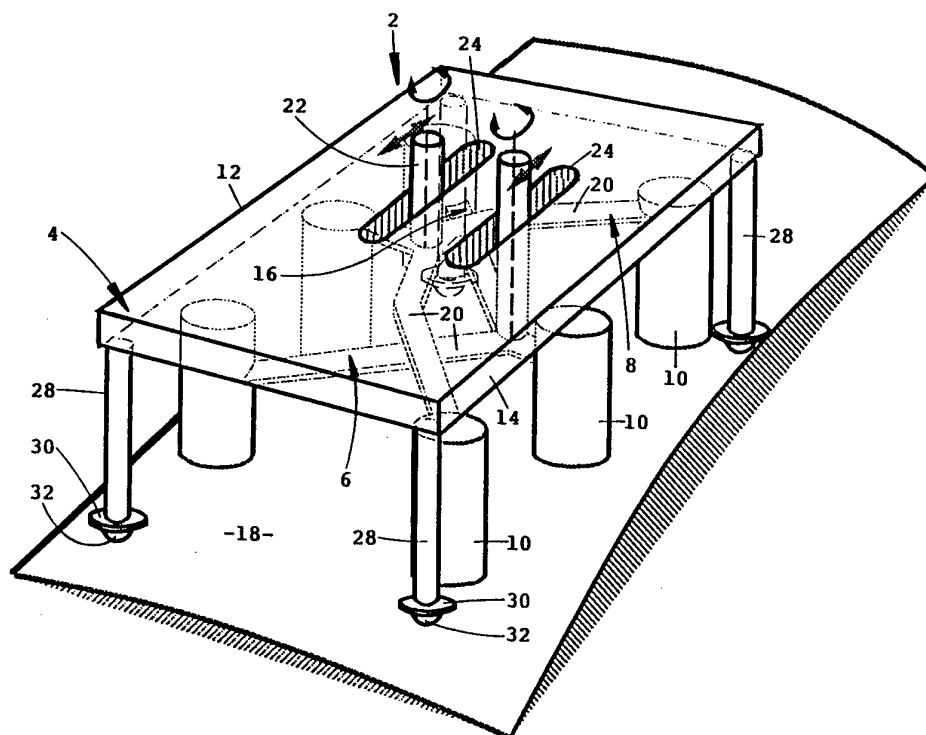




INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁵ : B25J 11/00, 18/02, B23Q 16/00	A1	(11) International Publication Number: WO 94/06602 (43) International Publication Date: 31 March 1994 (31.03.94)
(21) International Application Number: PCT/AU93/00481 (22) International Filing Date: 20 September 1993 (20.09.93) (30) Priority data: PL 4826 18 September 1992 (18.09.92) AU (71) Applicant (for all designated States except US): COMMON-WEALTH SCIENTIFIC AND INDUSTRIAL RE-SEARCH ORGANISATION [AU/AU]; Limestone Avenue, Campbell, ACT 2601 (AU). (72) Inventors; and (75) Inventors/Applicants (for US only) : GUO, Lin [AU/US]; 6009 Wenk Avenue, Richmond, CA 94804 (US). ROG-ERS, Kevin, John [AU/AU]; 90 Main Street, Blackburn, VIC 3130 (AU). KIRKHAM, Robin, John [AU/AU]; 16 Westbourne Grove, Northcote, VIC 3070 (AU). ROGO-ZINSKI, Maciej, Włodzimierz [AU/AU]; 10 Cuthbert Street, Bulleen, VIC 3105 (AU).		(74) Agents: WEBBER, David, Brian et al.; Davies Collison Cave, 1 Little Collins Street, Melbourne, VIC 3000 (AU). (81) Designated States: AU, CA, JP, US, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE). Published <i>With international search report.</i>

(54) Title: A ROBOT



(57) Abstract

A robot comprising a body (4), and two legs (6, 8) without active joints, said legs being mounted to said body such that the legs can be independently translated and rotated with respect to said body.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	FR	France	MR	Mauritania
AU	Australia	GA	Gabon	MW	Malawi
BB	Barbados	GB	United Kingdom	NE	Niger
BE	Belgium	GN	Guinea	NL	Netherlands
BF	Burkina Faso	GR	Greece	NO	Norway
BG	Bulgaria	HU	Hungary	NZ	New Zealand
BJ	Benin	IE	Ireland	PL	Poland
BR	Brazil	IT	Italy	PT	Portugal
BY	Belarus	JP	Japan	RO	Romania
CA	Canada	KP	Democratic People's Republic of Korea	RU	Russian Federation
CF	Central African Republic	KR	Republic of Korea	SD	Sudan
CG	Congo	KZ	Kazakhstan	SE	Sweden
CH	Switzerland	LI	Liechtenstein	SI	Slovenia
CI	Côte d'Ivoire	LK	Sri Lanka	SK	Slovak Republic
CM	Cameroon	LU	Luxembourg	SN	Senegal
CN	China	LV	Latvia	TD	Chad
CS	Czechoslovakia	MC	Monaco	TG	Togo
CZ	Czech Republic	MG	Madagascar	UA	Ukraine
DE	Germany	ML	Mali	US	United States of America
DK	Denmark	MN	Mongolia	UZ	Uzbekistan
ES	Spain			VN	Viet Nam
FI	Finland				

- 1 -

A ROBOT

The present invention relates to a robot for industrial use.

5 Most robots used in the manufacturing industry have a fixed base and carry a tool which the robot moves relative to the base in order to perform work on a particular part. Positional parameters for movement of the tool to perform work on the part are specified with respect to the base of the robot and the robot is preprogrammed to move the tool in accordance with the parameters. Parts are normally transported to the robot and
10 accurately positioned relative to the robot base in a specially designed fixture. Robots of this type have proved to be adequate for processing small and medium sized parts in relatively high numbers but become impractical and expensive as the size of the part increases. A robot often needs to operate over most of the surface of a part and as the size increases, inevitably the length of the robot's links increase, which increases the
15 complexity and cost of the robot. Also this makes it more difficult to precisely manipulate the tool and the reduced stiffness of the robot can introduce inaccuracies in positioning of the tool.

For large parts, such as storage tanks, buildings and ships, conventional robots are
20 unsuitable as they normally only have a reach up to 3 metres. Large parts are also difficult to transport and locate accurately with respect to a robot and if the parts are produced in small quantities, then preprogramming of a robot for each part becomes expensive and inefficient. Work on such parts would be better performed by a portable robot which can move over the part so as to perform the desired task. The task may
25 include welding, cutting, polishing, inspection or sealing and require continuous motion over the part. In a number of instances, the task may need to be performed relative to a feature of the part, such as a seam, joint or line, and the robot accordingly needs to track the feature, preferably without part specific programming, to perform the required operation.

30

Robots which have been developed to walk fall within one of three categories. The first includes robots with several legs, each with multiple joints. They are complex

SUBSTITUTE SHEET

- 2 -

and expensive and are used primarily for scientific, experimental or entertainment purposes, and are not practical or economic for industrial use. The second category comprises dynamic legged robots which are able to run and hop on a surface and rely on the dynamic exchange of energy between energy storage devices in the robot, such as springs, and the kinetic or potential energy of the robot to move. Robots of this second category have not yet been applied industrially and their motion is relatively erratic and jerky. The third category includes static legged robots which generally have only two legs, one leg fixed to a frame which carries the tool or a payload, and the second leg moveable relative to the frame. The robots of this type move across a part in a discontinuous or stop-start manner. The attitude of the frame with respect to the surface of the part also changes discontinuously as the robot moves across a curved surface. Discontinuous motion is undesirable for many tasks.

In accordance with the present invention there is provided a robot comprising a body, and two legs without active joints, said legs being mounted to said body such that the legs can be independently translated and rotated with respect to said body.

Advantageously said legs may each include at least one foot.

Preferably the feet are adapted to retract and extend relative to the respective legs to engage and disengage a surface.

Advantageously the robot may further include abutting means which extend from and set the attitude of said body, wherein said feet on engaging said surface are adapted to retract to ensure the ends of said abutting means abut said surface.

Preferably the ends of said abutting means are configured to move over said surface and the motion of the legs is such that, at any given time during movement of said robot over said surface, said at least one foot of at least one of the legs engages said surface and said ends substantially continuously abut said surface. Retraction of the engaging feet is such that contact is maintained between the surface and said ends.

SUBSTITUTE SHEET

- 3 -

Preferably said feet each include a surface engaging shoe having means for adhering to said surface. Preferably said shoe is pivotal within a body of the foot to facilitate tangential engagement of said shoe with said surface.

5 Advantageously the robot may include a tool mounted on said body and which is moveable relative to said body.

Preferably the robot includes means for providing position information to said robot, and means for controlling said moveable legs in response to said information.

10

Preferably said controlling means further controls movements of said tool relative to said body in response to the position information.

Preferably said information providing means comprises means for recognising a
15 feature of the surface and determining said position information for said feature relative to said robot, so as to cause said robot to track said feature.

Preferably the controlling means includes a pair of motors for each moveable leg which act on a shaft of the respective moveable leg, wherein uniform control of said
20 motors effects translational movement of said shaft and differential control of said motors effects rotational movement of said shaft relative to said body.

Preferably the feet further includes ram means for retracting and extending said shoes of said feet.

25

Advantageously said robot is able to move across said surface in a continuous motion. The robot is also able to maintain a substantially continuous attitude for said body with respect to said surface, even when said surface is curved.

30 Preferred embodiments of the present invention are hereinafter described, by way of example only, with reference to the accompanying drawings, wherein:

Figure 1 is a diagram of the basic components of a preferred embodiment of a

SUBSTITUTE SHEET

- 4 -

robot;

Figure 2 is a schematic diagram illustrating six phases of movement of the robot in a straight line;

Figure 3 are graphs of the velocities of parts of the robot during the six phases
5 of movement;

Figure 4 is a perspective view of mechanical components of the robot;

Figure 5A is a cross-sectional view of the toe of the robot in a retracted position;

Figure 5B is a cross-sectional view of the toe of the robot in an extended position;

10 Figure 5C is a cross-sectional view of the toe of the robot in an extended position with a shoe of the toe pivoted;

Figure 6 is a pneumatic circuit diagram of a pneumatic circuit for control of the toes of the robot;

Figure 7 is a plan of a frame and toe pattern for the robot;

15 Figure 8 are plans of three alternative toe arrangements;

Figure 9 is a graph of robot mobility envelopes for alternative toe arrangements;

Figure 10 is a graph of robot mobility envelopes for alternative shaft separations of the robot;

Figure 11 is a perspective view of a leg arrangement for the robot;

20 Figure 12 is a plan of a preferred leg arrangement;

Figure 13 is a block diagram of the modules of a preferred control system of the robot;

Figure 14 is a block diagram of a motion control system of the robot;

Figure 15 is a diagram of a robot coordinate system for the control system;

25 Figure 16 is a diagram of movement of tracking and directional points in the robot coordinate system;

Figure 17 is a diagram of a tracking point path and a predicted tool point path traversed during tracking of a feature by the robot;

Figure 18 is a diagram of joint trajectory and setpoint points for a smooth joint
30 trajectory of the robot; and

SUBSTITUTE SHEET

- 5 -

Figure 19 is a block diagram of the electrical components of the preferred control system of the robot.

A robot 2, as shown in Figure 1, includes a rectangular frame 4, and first and second legs 6 and 8. The legs 6 and 8 each include three respective toes 10. The toes 10 are disposed underneath the frame 4 and for each leg 6 and 8 are positioned to define points of a triangle parallel to the plane of the frame 4. The base of the triangle formed by the toes 10 of the first leg extends along one side 12 of the frame 4 and the base of the triangle formed by the toes 10 of the second leg 8 extends along the opposite side 14 of the frame 4. The toe 10 which forms the apex of each triangle is disposed in the middle of the first side 12 for the second leg 8 and in the middle of the other side 14 for the first leg 6. The centroid of all of the toes 10 relative to a plan view plane of the frame 4 is approximately the centre 16 of the frame 4, and will move as the legs 6 and 8 move. The toes 10 are used to engage the surface 18 of a part over which the robot 2 moves, and the frame 4 is positioned over the surface 18 and the toes 10. The toes 10 of each leg 6 and 8 are connected by respective struts 20 which extend between the toes 10 of each leg 6 and 8, as shown in Figure 1. The struts 20 of the first leg 6 are disposed below the struts 20 of the second leg 8. The struts 20 for a leg 6 or 8 meet at a point near the toe 10 which forms the apex of the triangle defined by the toes 10 of the leg. The legs 6 and 8 each include a shaft 22 which extends upwardly from the intersecting point of the struts 20. The shafts 22 extend up through respective longitudinal slots 24 in the frame 4, each slot 24 being disposed on respective sides of and equidistant from the centre 16 of the frame 4.

Movement of the frame 4 and legs 6 and 8 relative to one another is controlled by equipment, described hereinafter, mounted on the frame 4 which acts on the shafts 22. Movement of the shafts 22 is constrained by the equipment to rotation about their respective axis and translational movement in one plane along the respective slot 24. The legs 6 and 8 therefore have two degrees of freedom. The shafts are rotated and translated independently so as to move the legs 6 and 8 and the robot 2 across the surface 18, as desired. The shafts 22 are not raised and lowered to engage or disengage the surface 18 but instead the toes 10 of a leg 6 or 8 are retracted or extended.

- 6 -

The robot 2 further includes four legs 28 fixed to each corner of the frame and which each extend down from the frame 4. The lower ends 30 of the legs 28 include a wheel or a roller 32 which abuts and moves across the surface 18. The legs 28 define the attitude of the frame 4 with respect to the surface 18. On engaging the surface 18, the toes 10 of a leg 6 or 8 are retracted so as to maintain contact between the surface 18 and the rollers 32 of the legs 28. During movement of the robot 2 across the surface 18, at any given time the toes 10 of at least one leg 6 or 8 engage the surface 18 and retraction of the engaging toes 10 ensures the rollers 32 maintain contact with the surface 18. The attitude of the frame 4 remains the same with respect to the surface 18 and is therefore continuous as the robot 2 moves across the surface 18. The phases of a walking cycle when the robot 2 travels in a straight line are illustrated in Figure 2. A first graph 40 of the velocity of the frame 4 relative to the surface 18 over each phase is shown in Figure 3, together with a second graph 42 of the velocities of the legs 6 and 8 over the phases, and a third graph 44 of the velocities of the motors which respectively drive the legs 6 and 8 over the phases. In the second and third graphs 42 and 44, the plot for the first leg 6 and its respective motor is given by the solid lines 46 and 48 whereas the plots for the second leg 8 and its respective motors are given by the dotted lines 50 and 52. As is apparent from the graphs 40, 42 and 44, the frame 4 maintains a constant velocity V_r with respect to the surface 18 over all of the walking cycle phases. Movement of the legs during each of the phases is as follows:

1. The first and second legs 6 and 8 both engage the surface 18, and the motors for each leg rotate at W_1 to drive the frame forward at velocity V_r with respect to the surface 18. The frame 4 can only have straight line motion when both legs 6 and 8 engage the surface 18 as angular motion is not possible because the centres of both shafts 22 are not coincident.
2. The toes 10 of the first leg 6 are retracted from the surface 18, and the motor velocities remain the same.
3. The frame 4 continues to be driven forward at V_r by the motors of the second leg 8. The first leg 6 is driven forward at a high velocity $V_{reset} > V_r$ until it reaches

SUBSTITUTE SHEET

- 7 -

a reset position in front of said second leg and its motors are stopped.

4. The frame continues to be driven forward at V_r by the motors of the leg 8 and the first leg 6 moves at V_r with the frame 4.
5. The motors of the first leg 6 are driven so the first leg 6 is made stationary with respect to the surface 18. The motors of the second leg 8 continue to drive the frame forward at the velocity V_r .
- 10 6. The toes 10 of the first leg are extended to engage the surface 18 in front of the toes 10 of the second leg 8. One step of the walking cycle is completed and the six phases are repeated for the next step with the roles of the first leg 6 and the second leg 8 being reversed. Any number of successive steps can be executed to move the robot over a part.

15

The shafts 22 of the robot 2, as shown in Figure 4, are each held within two aligned cylindrical bearings 60 which extend from the side of a first nut 62. The nuts 62 are mounted on tracks 64 of the frame 4 which extend longitudinally with respect to the frame. One track 64 for each nut 62 may be used as illustrated, or alternatively a pair of tracks 64 can be used for each nut 62 to enhance the mechanical stiffness of the structure. The tracks 64 effectively define the two slots 24. The nuts 62 are each driven along the respective track 64 by a first ball screw 66 which lies parallel with the track 64 and is driven by a first servo motor 68. A second servo motor 70 is mounted on the outside of the first servo motor 68 for each of the shafts 22, and the second servo motor 25 70 drives a second ball screw 72, which is parallel to the first ball screw 66. The rotation of the second ball screw 72 drives a second nut 74 along the screw 72. An arm 76 extends over the first nut 62 and connects the second nut 74 to the respective shaft 22. One end of the arm 76 is fixed to the head 78 of the shaft 22 and the other end is pivotally mounted on the second nut 74. Translational movement of a shaft 22 is 30 achieved by driving the respective pair of motors 68 and 70 in unison, i.e. in the same direction, and the translational movement is confined by the track 64. Differential movement of the first and second nuts 62 and 74 causes the arm 76 to pivot about its

SUBSTITUTE SHEET

- 8 -

mounting on the second nut 74 and forces the shaft 22 to rotate within the bearings 60. Rotational movement of a shaft 22 is therefore controlled by differential rotation of the pair of motors 68 and 70.

- 5 Pneumatic cylinders 80 are mounted on each of the toes 10 to retract and extend the toes 10 independently. The ends 30 of the fixed legs 28 each include rollers 32.

A toe 10, as shown in Figures 5A to 5C, is bolted to a leg 6 or 8 underneath a respective pneumatic cylinder 80. The toe 10 includes a toe body 82 having an open end
10 which receives an open ended cup 84. The open end of the cup 84 receives a shoe 86 of the toe 10. The closed end 88 of the cup 84 is pushed downwards by a washer 94 attached to the piston 90 by a bolt 92, when the piston 90 is extended. The shoe 86 has a partly spherical convex outer surface 100 which abuts the rim of the cup 84, and the shoe 86 is held in the cup 84 by a tensile link 96. The link 96 has spherical ends that
15 are received by the bolt 92 and a connecting part 98 on the top of the shoe, respectively. Contact is maintained between the surface 100 and the cup 84 by three springs 102 connected between the top of the shoe 86 and the closed end 88 of the cup 84. The link 96 allows the shoe 86 to be retracted by the cylinder 90, which also causes the cup 84 to be retracted within the toe body 82. When the shoe 86 is retracted, a clearance 105,
20 as shown in Figure 5A, exists between the washer 94 and the top of the closed end 88 of the cup 84, and the link 96 is in tension. The clearance 105 and the spherical ends of the link 96 allow the tension in the link 96 to be removed and the shoe 86 to pivot within the cup 84 when the washer 94 is pressed against the closed end 88 during extension of the piston 90. The clearance 105 decreases as the shoe 86 pivots. The
25 curved outer surface 100 of the shoe 86 also facilitates the pivotal movement of the shoe 86. Three pins 104 extend from the top 108 of the toe body 82 through the closed end 88 of the cup 84 to abut the top of the shoe 86 when it is retracted into the body 82. The pins 104 reset the orientation of the shoe 86 so that its surface engaging face 106 is parallel with respect to the closed ends 88 and 108 of the cup 84 and the body 82. The
30 end of the pin 104 also includes a flange 110 which engages the end 88 of the cup 84 when fully extended by the piston 90 of the cylinder 80.

SUBSTITUTE SHEET

- 9 -

The ability of the shoe 86 to pivot within the cup 84, as shown in Figure 5, is particularly advantageous. The shoe 86 rotates about a centre of rotation which is below the surface 18 which the shoe 86 engages. This configuration ensures that when contact between the shoe 86 and the surface 18 occurs, the shoe is able to rotate rather than jamb
5 with contact only on one edge. The shoe 86 is therefore always able to achieve a tangential contact with the surface 18. Rotation of the shoe 86 within the cup 84 also allows the toes 10 to maintain the initial tangential contact with the surface 18 as the frame is moved with respect to the surface when the toes 10 are magnetically clamped to the surface 18 during a walk cycle.

10

The shoe 86 includes an annular hollow 112 in which a coil is wound about a central core pole 114. The coil 37 can be energised to magnetise the shoe 86 as an electromagnet. The robot 2 uses electromagnetic toes 10 for operation on ferromagnetic surfaces which may be hot, for example, from welding or cutting. The coil is shaped to
15 maximise turns while minimising local magnetic saturation in the shoe and this is achieved because the cross-sectional area of the magnetic path from the central pole 114 to the outer pole 116 is approximately constant. The surface engaging face 106 is formed primarily by a pole piece 118 which is inserted into the central pole 114. The pole piece 118 increases the circumference of the central pole 114 and allows the robot 2 to operate
20 on surfaces of parts which are relatively thin without magnetic saturation. The pole piece 118 also retains the coil in the shoe 86.

To confirm attachment of the toes 10 to a surface 18, an electronic circuit (not shown) may be used to monitor the change in inductance of the coil in the shoe 86 when
25 the shoe contacts a ferromagnetic surface. The circuit superimposes a relatively small low frequency AC signal on the DC signal supplied to energise the coil. The phase shift between the superimposed AC signal and the monitored AC voltage of the coil is digitally timed and compared with a threshold value. A leg 6 or 8 is only considered to be secured to the surface 18 when the phase shift for each of the coils of the leg's three
30 toes 10 exceeds the threshold value. The circuit may also be used to monitor the inductance of the coil 37 so as to determine when the respective shoe 86 abuts the surface 18 and the coil can be energised to secure engagement. The toes 10 can,

SUBSTITUTE SHEET

- 10 -

however, also simply be energised after the respective cylinder 80 has been extended the full length, which may take 150 ms.

When the coil 37 is de-energised, residual magnetic fields can cause a toe 10 to
5 remain adhered to the surface 18 with a force less than that generated when the coil 37 is energised. This can induce shocks on release of the toes 10 and affect the quality of the motion of the robot 2 so a reverse current is applied to the coil 37 for a short period of time to reduce the residual magnetic fields to a negligible level.

10 A pneumatic circuit 120 for controlling pneumatic cylinders 80 of the legs 6 and 8 is shown in Figure 6. The pressure in the upper chamber 122 of the cylinders 80 is maintained constant, typically 1 bar. The pressure in the bottom chamber 124 is atmospheric pressure when the cylinder piston 90 is extended. When a toe 10 engages
15 a surface 18, the pressure in the lower chamber 124 is set to a higher value depending on the weight and attitude of the robot 2, and other external forces. The higher pressure setting ensures the toe 10 pulls the respective leg 6 or 8 towards the surface 18 with a predetermined force independent of the length of extension of the toe 10. To extend the pistons 90 of the cylinders 80, respective solenoid valves 128 are de-energised, as shown in Figure 6, to vent the lower chambers 124. After the shoes 86 have contacted the
20 surface 18, the coils of the shoes 86 are energised and the lower chambers 124 are pressurised to the desired higher value. The frame 4 is pulled toward the surface 18 and contact between the wheels 32 and the surface 18 is maintained. When the coils are de-energised, the pressure in the lower chambers 24 causes the cylinders 80 to retract the shoes 86. The valves 128 are inexpensive compact three-way valves which the circuit
25 122 uses to achieve two-way force control of the cylinders 80. A regulator 130 sets the pressure in the upper chambers 122, and therefore the extension force of the cylinders 80. An electronically set regulator 132 may be used to facilitate control of the cylinder pulling force in response to operational factors including the attitude of the robot 2 with respect to horizontal. The circuit 122 also includes for each leg 6 and 8 a differential
30 pressure switch 134, a non-return valve 136 and a two-way valve 138 to vent the upper chambers 122 of each leg when their pressure exceeds the setting of the regulator 130 by a predetermined value, for example 0.2 bar. This overcomes any inability of the regulator

- 11 -

130 to vent flow arising from retraction of the cylinders 80.

The force between a shoe 86 and the surface 18 is the difference between the magnetic force and the pulling force exerted on the shoe 86 by the respective cylinder 80. Traction of the robot 2 is maximised by maximising the shoe to surface force, which requires that the cylinder pulling force be a minimum that is consistent with maintaining contact between the wheels 32 and the surface 18. This also reduces the contact force between the wheels 32 and the surface 18 and improves the ability of the robot 2 to travel over irregularities, such as bumps, of the surface 18.

10

The mobility of the robot 2 is constrained by the limitations on the strokes of the nuts 62 and 74 on the ball screws 66 and 72 and by interference between components of the robot 2 during movement. Computer simulations have been performed to select a design which minimises interference. For example, if the robot 2 has dimensions for the frame 4 and legs 6 and 8, as shown in Figure 7, then three possible toe arrangements A, B and C could be used, as shown in Figure 8, where the centres of the shafts 22 of legs 6 and 8 are O_1 and O_2 , respectively, and are chosen to be 100 mm apart. A plan view pattern for the wheels 32 and toes 10 is selected to provide the longest first step, initially considering only toe/toe and toe/wheel interference. A graph 130 of the maximum step length l_1 achievable for different orientations, α , of the frame 4 relative to the clamped or engaging leg 6 or 8 is shown in Figure 9 for each of the alternative toe arrangements A, B and C. From the graph 130 it is apparent that arrangement A offers the best mobility unless rotations of the robot 2 are extremely important for a given task.

25 The separation of the two shafts 22 is determined after considering multiple steps of the robot 2 and the graph 132 is shown in Figure 10 for the results obtained for four different separations S . Positive separations occur for the arrangement A as shown in Figure 8 and negative values for S occur when the orientation of the centres of the shafts 22 are reversed. From the graph 132 it is apparent that small shaft separations increase mobility, and negative separations give a mobility plot or envelope which is more symmetric about $\alpha = 0$. For example, the separation $S = -100$ mm may be chosen as a reasonable minimum separation given anticipated sizes of the shafts 22.

SUBSTITUTE SHEET

- 12 -

The shape of the legs are designed to minimise interference, and Figure 11 illustrates a basic configuration for the legs 6 and 8 where interference is minimised with the first leg 6 arranged so the interconnecting struts 20 extend below those of the second leg 8. Therefore the only possible interferences, in addition to those previously
5 considered, are between the structure of the second leg 8 and the shaft 22 of the first leg 6, and between the struts for one leg 6 or 8 and the toes 10 of the other leg 8 or 6. For the selected toe and wheel pattern, Figure 12 shows the plan of a frame and leg assembly for the robot 2 which maximises mobility. Possible collision spots include first spots 140 between the front and back toes 10 of the first leg 6 and the structure of the second leg
10 8, and second spots 142 between the shaft 22 of the first leg 6 and the structure of the second leg 8.

The robot 2 includes an electrical control system 250, as shown in Figure 19, controlled by either first control software shown in Figure 13 or second control software
15 listed on pages 32 to 94. The robot 2 has a tool (obscured) mounted underneath the frame 4. The tool is used to perform work on the part over which the robot 2 moves. The first control software 200 is able to monitor a feature of the part so the robot 2 can track the feature and position the tool relative to the feature, as desired. The second control software positions the tool relative to the feature after receiving information
20 concerning the position of the feature relative to the frame 4.

The control system 250, as shown in Figure 19, includes DC servo motors 68, 70, such as the Electrocraft S19-1A and S586-1A, which drive ball screws 66 and 72 of the robot 2, respectively. The motors 68, 70 are each provided with a tachometer 254 and
25 an optical encoder 256, which is used to monitor the movement and position of a respective nut 62, 74. The motors 68, 70 are driven by respective servo drivers 258, such as the Galil SSA-8/80 which is provided with analogue velocity and current loops. The solenoid valves 128 and the magnetic coils 37 are powered by an I/O solenoid valve and magnet drive unit 260. The servo drivers 258 and the magnet and solenoid drive 260 are
30 both controlled by a controller 252 via a 30 metre long umbilical cable 262 which transmits electrical and pneumatic power to the robot 2. The umbilical cable 262 also returns the feedback signals from the encoders 256 to the controller 252, together with

SUBSTITUTE SHEET

- 13 -

signals from microswitches which are used to limit travel of the drive shafts of the servo motors 68, 70. Digital proportional integral derivative (PID) position control of the nuts 62 and 74 is performed by two Industry Pack servo control modules 274 produced by a GreenSpring Computers Inc., using the information provided by the encoders 256. The
5 servo control modules 274 include two channels and are based on the National Semiconductor LM628 chip. Two Industry Pack digital I/O modules 270 remotely control the servo drivers 258 and the magnetic and solenoid drive 260.

Control of the modules 270, 274 is provided by an MV147 central processing unit
10 266 via a VME (Versa Module Euro card) bus 268. The CPU 266 is used to run control software on the VxWorks real time operating system. Alternatively, the control software can be run off-line on Unix workstations connected to the CPU 266 for development and testing.

15 The controller 252 can be configured, if desired, for mounting on the robot 2.

The first control software 200, as shown in Figure 13, takes into account that the robot 2 may provide an additional axis of the tool movement transverse to frame 4. This allows the tool to have a more complex path and rapid short stroke local motions as may
20 be required to weave a welding torch during a welding operation. The stroke of the transverse tool axis is determined by the minimum radius of curvature on the surface 18 of the feature to be tracked and any additional motion related to the process to be performed, such as welding torch weave. For example, sharper feature curves increase the stroke of the tool axis as the robot moves inside a feature corner while it is moved
25 along a tracked feature. Process data 212 from the tool may also be used to provide data on the location of the feature relative to the tool. For example, in welding this is used as the basis for through-arc sensing for seam tracking systems of conventional robots.

The first control software 200 uses the optical sensor 202 which includes a light
30 source and video camera mounted on the frame 4. The optical sensor 202 captures an image of a light stripe, projected transverse ahead of the robot 2 and across the feature. The shape of the light stripe image indicates the location and cross-section of the feature.

SUBSTITUTE SHEET

- 14 -

The optical sensor 202 performs standard image processing techniques on the image to provide data on the location of the feature with respect to the robot and the feature dimensions to a motion control system (MCS) 204. Other feature sensors may be employed, such as mechanical probes and flux sensors. The MCS 204 also receives data
5 on the state of the robot, motion parameters and a desired traversal velocity which may be inputted by an external operator or process controller 206. On the basis of all the entered data, the MCS 204 determines the desired positions and rotations of the legs 6 and 8 and generates joint position commands which are fed to a joint servo system (JSS) 208.

10

The term joint as used herein refers to the interface between two parts which constrains relative movement between the parts to one type of motion (i.e., translation or rotation). The permitted motion is normally actively controlled by a servomechanism. The robot 2 has four actively controlled or active joints which position the legs 6 and 8
15 relative to the frame 4. The legs 6 and 8 each have two degrees of freedom relative to the frame 4 and each degree of freedom is controlled by an active joint. Each of toes 10 may be considered to include one passive joint which allows rotation of shoe 86 and one semi-passive joint relating to extension or retraction of the toe.

20

The JSS 208 includes standard hybrid or digital servo control mechanisms 274, 258, 254, 256 for controlling the motors 68 and 70 in response to the joint position commands. The JSS also supplies joint positions signals which provide information on the position of the legs 6 and 8. The MCS 204 further controls changeover of the legs 6 and 8 and determines the desired position of the tool along the tool axis and generates
25 a predicted tool position command for a tool control system (TCS) 210. The TCS 210 also receives data on the feature from the process data 212 discussed previously. The predicted tool position command corresponds to placing the tool over the feature position predicted from the joint position commands or joint position signals and the feature data obtained by the optical sensor 202.

30

The TCS 210 performs fine positioning of the tool over the feature, and with the feature data obtained from the process data 212, the TCS 210 is able to operate in a

SUBSTITUTE SHEET

- 15 -

closed-loop mode to reduce errors in tool position. The TCS 210 generates a tool position command for a tool servo system (TSS) 212 which includes a standard hybrid or digital servo mechanism for controlling the motor which drives the tool transverse to the frame 4. When the process data 212 is not available, the TCS simply passes the
5 predicted tool position command as the tool position command to the TSS 212.

The motion parameters used by the MCS 204 are maximum velocities and accelerations for the motors of the robot 2, a joint clock interval T_j (the sampling period of the JSS 208), a measurement period T_r of the optical sensor 202, a joint trajectory
10 smoothing interval T_s discussed hereinafter, and the time required for engaging and disengaging the toes 10 from the surface 18. Except the latter, all of the time intervals are multiples of the joint clock interval T_j . The MCS 204 and the TCS 210 are processing systems implemented in software on the CPU 266 of the robot 2, which preferably runs under a real time operating system such as VxWorks. The joint positions
15 and the positions of the tool along the tool axis are sensed conventionally, for example, by the incremental encoders 256. A path planning module can replace the optical sensor 202, if the robot 2 is required to execute a predetermined motion with respect to the surface rather than track a feature, and the module would provide data of a similar form to the MCS 204.

20

The MCS 204 comprises four software modules, as shown in Figure 14, a supervisor module 220, a feature trajectory generator (FTG) 222, a joint setpoint generator (JSG) 226, and a joint trajectory generator (JTG) 228, which are described
hereinafter.

25

For feature tracking, two points on the frame 4 are defined:

1. A tracking point, which is the point the MCS 204 attempts to move towards the measured feature point in such a way that the feature remains within the range of
30 the optical sensor 202. The tracking point shifts along the line which is perpendicular to and bisects an optical sensor detection line, which corresponds to the light stripe shone across the feature by the optical sensor 202. As the

- 16 -

feature deviates from the centre of the sensor detection line, the tracking point shifts towards the sensor detection line to extract movement from the robot 2 to keep the feature in view of the sensor. If the feature is near the centre of the sensor detection line, the tracking point shifts away from the sensor detection line, towards the tool axis. In this case, the centre of the tool axis is positioned close to the feature, and tracking is more closely aligned with the tool. The axis of the tool is preferably located below the motors 68 and 70 at the front of the robot 2. The tracking strategy relies on the tracking point being in front of the centre of the shaft 22 of the clamped leg 6 or 8 at all times.

10

2. A predicted tool point, which is the intersection of the tool axis and the predicted path of the feature. It is desirable to keep this point near the centre of the tool axis to minimise the axis stroke and interference between the tool and other robot components. This is achieved when the tracking point is shifted towards the tool axis.

15

The position of the tracking point is determined by the supervisor 220 which seeks a compromise between the usually conflicting objectives of keeping the feature within the sensor detection line and keeping the tool close to the centre of the tool axis.

20

The robot coordinate system (RCS) is defined as having its origin at the centre of the shaft of the clamped leg (O_1 or O_2 in Figure 8) and its x axis parallel to the longitudinal axis of the frame 4. In the RCS, the tracking point has coordinates x_t , y_t and a feature point has coordinates x_f , y_f , as shown in Figure 15. The y_t coordinates are determined by the optical sensor 202 and adjusted for whichever leg is currently clamped. The x_t coordinates result from movement of the shaft of the clamped leg along the frame 4. Feature measurements are performed as the robot moves and coordinates in the RCS of a sequence of feature points are created by the FTG 222. These are made available for the joint setpoint generator 226, as shown in Figure 14.

25
30

Selection of the feature sensing interval T_f depends on the tool positioning requirements and the feature radius. Decreasing T_f improves the accuracy and allows

SUBSTITUTE SHEET

- 17 -

paths of smaller radius to be tracked.

The method of generating the joint trajectories, as described hereinafter, requires at least two points of the feature, which implies that $d_{st} \geq 2V_r T_f$, where d_{st} is the
 5 x-direction distance between the sensor detection line and the tracking point and V_r is the transversal velocity.

The joint setpoint generator (JSG) 226 computes angles and distance setpoints such that the rotation and translation of frame with respect to the clamped leg moves the
 10 tracking point towards the next feature point while maintaining constant tool traversal velocity.

The angle between the x axis and the tracking point (x_t, y_t) is denoted θ (positive anticlockwise), as shown in Figure 16. The directional point (x_d, y_d) results from the
 15 following procedure:

1. The coordinates of the feature point closest to the tracking point are established;
2. If the distance between the tracking point and this feature point is smaller than the
 20 distance $d_t = V_r T_f$, the next closest feature point is selected and the procedure repeated;
3. The coordinates x_d and y_d of the point lying at distance d_t along the line connecting the tracking point and the selected feature point are calculated.

25

The translational and rotational setpoints for the joints of the clamped leg required to move the tracking point from its present position (x_t, y_t) towards the directional point (x_d, y_d) are calculated as follows. Firstly, the rotation Φ is calculated:

$$\phi = -2 \arctan \left(\frac{x_d - (x_d^2 + y_d^2 - x_t^2 (1 + \tan^2 \theta) \sin^2 \theta)^{1/2}}{y_d + x_t^2 (1 + \tan^2 \theta)^{1/2} \sin \theta} \right) \quad (1)$$

- 18 -

Next, the translation distance D is calculated. If the tracking point coincides with the predicted tool point (x_p, y_p) , then

$$D = - \frac{(x_t^2 (1 + \tan^2 \theta))^{1/2} \sin (\Phi + \theta) - y_d}{\sin \Phi} \quad (2)$$

Otherwise, D is the distance between the predicted tool point, after rotation by Φ , and a point identified as follows. The point is the intersection of the circle $(x - x_p)^2 + (y - y_p)^2 = d_t^2$ and the line $y = mx + c$ which has the greater x coordinate. The line parameters are

$$m = \tan^{-1} \Phi \text{ and } c = \sqrt{x_p^2 + y_p^2} (\sin(\theta + \Phi) - m \cos(\theta + \Phi))$$

The calculation above ensures that the tracking point moves towards the directional point, as shown in Figure 16. The value of D guarantees that the predicted tool point is moved by the distance d_t , and thus ensures constant traversal velocity of the tool. If D is evaluated according to (2), the tracking point moves to the directional point; otherwise it moves towards it.

The joint trajectory generator (JTG) 228 produces joint trajectories (translational and angular movement commands) for feature tracking, leg velocity matching (for leg changeover), leg halting, and leg resetting. Furthermore, the JTG 228 computes the predicted tool position command to position the tool along the tool axis over the expected feature location.

In Figure 17, the dotted line represents the feature to be tracked. The points Z_{feature} , B_{feature} , C_{feature} and D_{feature} are the points at which the feature is measured at intervals of T_f by the optical sensor while the robot moves. In the RCS, these points constitute the feature trajectory produced by the FTG 222.

On request from the JTG 228, which issues an evaluate setpoints signal as shown in Figure 14, the JSG 226 computes the joint setpoints Φ and D based on the feature trajectory produced by the FTG 222. The joint setpoints are generated within the period T_f unless there is a leg changeover operation, discussed hereinafter. The JTG 228

- 19 -

computes joint trajectories for transition between subsequent joint setpoints.

In order to maintain continuity of motion, the following preview joint trajectory generation method is used and is referred to as the preview joint trajectory generation method because the coordinates of the point C_{feature} are needed before the motion segment aiming at point B_{feature} is completed.

The method maintains continuity of joint position and velocity by smoothing the transition between subsequent segments of the joint trajectory j starting at time $t = -t_s$, as shown in Figure 18, where the smoothed (rotational and translational) joint trajectory generated at $t = -T_f$ is a bold dotted line and at $t = -t_s$ becomes a bold solid line. In Figure 18, $j_s^{(n)}$ denotes the joint setpoint (Φ or D) required to move the tracking point from point T to point S, the coordinates of which are expressed in the RCS associated with point T; j_s denotes the joint trajectory value when the tracking point is at point S.

The time interval t_s is referred to as the joint trajectory smoothing interval.

A third degree polynomial $q(t) = a_3t^3 + a_2t^2 + a_1t + a_0$ is used to smooth the trajectory $j(t)$ in the interval from $-t_s$ to 0, and a fifth degree polynomial $p(t) = b_5t^5 + b_4t^4 + b_3t^3 + b_2t^2 + b_1t + b_0$ is used to smooth the trajectory in the interval from 0 to t_s ; straight line interpolation is used from t_s to $T_f - t_s$.

The information needed at time $t = -t_s$ is j_A and $j_B^{(Z)}$ for the translational and rotational joints of the clamped leg. The following calculations are performed at time $t = -t_s$, i.e. when the tracking point is at point A.

25

1. Using j_A and $j_B^{(Z)}$ for each joint, the coordinates of the point C_{feature} are transformed from the RCS associated with point A, as shown in Figure 17, to the RCS associated with point B.

- 20 -

2. The joint setpoints, $j_C^{(B)}$, required to move the tracking point from point B towards point $C_{feature}$ are computed using coordinates of $C_{feature}$ in the RCS associated with point B. These setpoints would shift the tracking point from B to C.
- 5 3. The joint trajectories $j(t)$ are computed from time $t = -t_a$ to 0; this gives $j_{B'} = j(0)$ and the execution of these trajectories moves the tracking point to point B'.
4. The coordinates of the point $C_{feature}$ in the RCS associated with point A are transformed to the RCS associated with point B'.
- 10 5. The joint setpoints, $j_C^{(B')}$, required to move the tracking point from point B' towards point $C_{feature}$ are computed using the coordinates of $C_{feature}$ in the RCS associated with point B'. These setpoints move the tracking point to the final point C'.
- 15 6. The joint trajectories $j(t)$ from time $t = 0$ to $T_f - t_a$ are evaluated.

At time $t = T_f - t_a$ the following assignment is performed and the above steps repeated:

$$\begin{aligned}
 t &:= t - T_f \\
 j_A &:= j_X - j_B^{(Z)} \\
 j_B^{(Z)} &:= j_{B'} + j_{C'}^{(B')} - j_B^{(Z)} \\
 B_{feature} &:= C_{feature} \\
 C_{feature} &:= D_{feature} \\
 B &:= C' \\
 A &:= X \\
 Z &:= B'
 \end{aligned}$$

- 20 The algorithm is initialised after robot start-up and after each leg changeover. After initialisation, the trajectory from time $-T_f$ to $-t_a$ has a trapezoidal velocity profile (constant acceleration segment followed by constant velocity segment) as indicated by the thick dotted line in Figure 18.

- 25 The smoothed joint trajectory is evaluated at steps 3 and 6 of the above algorithm. The coefficients of the polynomial $q(t)$ are found as solutions to the equations:

SUBSTITUTE SHEET

- 21 -

$$\begin{aligned}
 q(-t_a) &= j_A \\
 \dot{q}(-t_a) &= -\frac{\Delta j_B}{t_a} \\
 \dot{q}(t_a) &= \frac{j_C^{(B)}}{T_f} \\
 \ddot{q}(-t_a) &= 0
 \end{aligned}$$

where $\Delta j_B = j_A - j_B^{(Z)}$. The coefficients are:

$$\begin{aligned}
 a_0 &= \frac{t_a j_C^{(B)} - 11T_f \Delta j_B + 12j_A T_f}{12T_f} \\
 a_1 &= \frac{t_a j_C^{(B)} - 3T_f \Delta j_B}{4t_a T_f} \\
 a_2 &= \frac{t_a j_C^{(B)} + T_f \Delta j_B}{4t_a^2 T_f} \\
 a_3 &= \frac{t_a j_C^{(B)} + T_f \Delta j_B}{12t_a^3 T_f}
 \end{aligned}$$

The coefficients of the polynomial $p(t)$ are solutions to:

$$\begin{aligned}
 p(0) &= q(0) \\
 p(t_a) &= \dot{p}(t_a) t_a + j_B^{(Z)} \\
 \dot{p}(0) &= \dot{q}(0) \\
 \dot{p}(t_a) &= \frac{\Delta j_C}{T_f} \\
 \ddot{p}(0) &= \ddot{q}(0) \\
 \ddot{p}(t_a) &= 0
 \end{aligned}$$

where $\Delta j_C = j_B + j_C^{(B)} - j_B^{(Z)}$. The coefficients are $b_0 = a_0$, $b_1 = a_1$, $b_2 = a_2$,

$$\begin{aligned}
 b_3 &= \frac{-120T_f j_A + 120T_f j_B^{(Z)} - 37t_a j_C^{(B)} + 155T_f \Delta j_B + 72t_a \Delta j_C}{12t_a^3 T_f} \\
 b_4 &= -\frac{19T_f \Delta j_B - 15T_f j_A + 15T_f j_B^{(Z)} - 4t_a j_C^{(B)} + 8t_a \Delta j_C}{t_a^4 T_f} \\
 b_5 &= 3 \frac{5\Delta j_B T_f - 4j_A T_f + 4j_B^{(Z)} T_f - j_C^{(B)} t_a + 2\Delta j_C t_a}{2t_a^5 T_f}
 \end{aligned}$$

- In Figure 17, the tracking point path deviates from the straight lines connecting
- 5 points Z, A and B' because the motion planning between them is performed in the joint rather than feature (Cartesian) space. This does not affect the predicted tool position. Trajectory smoothing in the joint space is desirable to reduce tendencies for toe-work slip, drive wear and robot vibration.

- 22 -

The smoothed trajectory for the translational joint consists of absolute linear displacement $j_{\text{trans lin}}$ which increases with forward robot motion. The smoothed trajectory for the rotational joint consists of absolute angular displacements $j_{\text{rot ang}}$ which increase for anticlockwise movement of frame about the shaft of the clamped leg. The drive
 5 arrangement employed to achieve rotation requires conversion of the rotational trajectory $j_{\text{rot ang}}$ to a translational trajectory $j_{\text{rot lin}}$ as follows:

$$j_{\text{rot lin}}(t) = j_{\text{trans lin}}(t) \pm r \tan(\phi(t))$$

where $\phi(t) = j_{\text{rot ang}}(t) - j_{\text{rot ang}}(t - T_j)$, T_j is the joint trajectory update period and r is the distance between the centres of the ball screws for translation and rotation. The positive sign is used for the leg whose shaft is at the right side of frame and the negative sign is
 10 used for the other leg.

At each clock interval $t = kT_j$, the feature trajectory is recalculated for the new RCS resulting from rotation by $\phi(t)$ and/or leg changeover.

15 Leg changeover involves matching the leg velocities, leg change-over, halting the lifted leg and resetting it to its foremost position and initial orientation.

During leg velocity matching, the stationary lifted leg, which is at its foremost position, has to be accelerated to the traversal velocity with respect to the frame.
 20 Meanwhile, the rotational velocity of the frame about the clamped leg is decelerated to zero and the translational velocity of the frame with respect to the clamped leg is changed to the traversal velocity, as discussed previously.

The joint position commands generated by the JTG 228 for leg changeover have
 25 trapezoidal velocity profiles. Each motion segment is completed at a multiple of the joint clock interval T_j .

The predicted tool position command is generated with the joint clock interval T_j , as the y coordinate of the intersection point of the tool axis with the line connecting the
 30 feature trajectory points immediately ahead and behind the tool axis. If the location of

SUBSTITUTE SHEET

- 23 -

the feature under the tool axis is accurately predicted, the tool is positioned at times $t = kT_j$ on the straight lines connecting subsequent feature points (Z_{feature} , B_{feature} , C_{feature} in Figure 17.) The predicted tool point follows the feature closely with a straight line approximation of the feature trajectory in "world" coordinates.

5

The supervisor module 220 supervises and controls tasks performed by other modules of the MCS 204. In particular, the supervisor 220 determines the tracking point, controls leg changeover, and adjusts the traversal velocity.

10 The tracking point (x_t , y_t) is located on the line bisecting the tool axis and the optical sensor detection line. Whenever the joint trajectories are generated, x_t is determined by:

$$\begin{aligned} x_t(t) &= x_w(t) + d_{sw} - [n(t)] V_r T_f \\ n(t) &= ([d_{sw} / (V_r T_f)] - 2) (1 - u(t)) + 2 \\ u(t) &= \begin{cases} u'(t) & \text{if } u'(t) \leq 1 \\ 1 & \text{otherwise} \end{cases} \\ u'(t) &= \frac{1}{e_{mx}} |k_p e(t) + k_d \Delta e(t)| \\ e(t) &= y_f(t) - y_s(t) \\ \Delta e(t) &= e(t) - e(t - T_f) \end{aligned}$$

where $x_w(t)$ is the x coordinate of the tool axis, d_{sw} is the x distance between the sensor detection line and the tool axis, e_{mx} is half the length of the sensor detection line, k_p and k_d are adjustable parameters, $y_f(t)$ and $y_s(t)$ are the y coordinates of the feature at the sensor detection line and of the middle of the sensor detection line, respectively, and $[x]$ denotes the integer part of x. This algorithm shifts the tracking point towards the sensor detection line when the feature moves out from the middle of the sensor, and towards the tool axis when the feature moves towards the middle of the detection line. The algorithm is a proportional-derivative type with positive k_p and k_d weighting the "proportional" and "derivative" terms, respectively.

The supervisor 220 also controls leg changeover. At each joint clock interval T_j , the supervisor evaluates the time t_{leg} required to complete leg velocity matching, leg changeover, and to halt the lifted leg. A safety margin (minimum one joint clock interval) is added to this to determine t_{leg} . If the motion commanded by the joint

- 24 -

trajectories during t_{leg} cannot be fully executed due to ball-screw limits or mobility constraints, the supervisor 220 issues the leg change command.

The desired traversal velocity may be set by the operator or by process
5 considerations. The supervisor determines the traversal velocity, V_r , used to generate joint setpoints. If the tracking point is near the optical sensor and the feature moves out of the sensor detection zone, the traversal velocity can be reduced to decrease the distance between the measured feature points. This reduces the radius of the feature which the robot can track.

10

The second control software is a simplified implementation of the first control software 200, in that feature sensing is omitted, and, as there is no feature measurement, a feature trajectory generator 222 is not required. Movement of the tool is also restricted to a simple engage and disengage operation and therefore an elaborate
15 tool control system 210 is not required. The second control software runs continuously to move a point on the robot, such as the position of the tool, along a path to a target point, the vehicle frame coordinates of which are either provided by a command or a remote joystick 280 to the controller 252.

20 The second control software comprises a number of C procedures, discussed hereinafter, listed on pages 32 to 85. Typical header files which define system parameters, variables and functions for the procedures are listed on pages 86 to 94.

On initially providing power to the robot 2, its state is determined and the servo
25 motors 68, 70 are calibrated using the microswitches. A "null" move to the target point is performed by making the target point coincident with the tool's initial position. The robot only begins to move when it is given a different target point. On receiving a new target point, positions for the servo motors 68, 70 are determined using a trajectory generator which plans motion of the legs 6 and 8 within mobility constraints, and controls
30 the legs 6 and 8 during the changeover phases. The trajectory generator is called every servo interval, which is approximately 20 ms, and is able to accept a new target point at any time except during leg changeover. If a new target is not provided, the existing

SUBSTITUTE SHEET

- 25 -

target point is used as a basis for moving the robot. The target point includes, in addition to the coordinates for a path point, the desired speed of travel of the tool or tool point. On accepting a target point, the trajectory generator calls an inverse kinematic module to determine the desired position and orientation of the shafts 22 required to move the robot 2 towards the target point. A range checking module is then called to determine whether the selected shaft positions and orientations are within the mobility constraints of the robot 2. If the constraints are not met, a closer target point is chosen midway between the unachievable target point and the current robot position. The process is repeated until the desired shaft positions and orientations are within the mobility constraints. Once this has been achieved, the number of servo intervals required to reach the target point is calculated from the desired speed and the servo motor positions required. At each servo interval the number of counts for each encoder 256, which need to be received to indicate that the desired nut positions have been reached, is determined and movement of the robot 2 effected. When the number of servo intervals remaining before the target point is reached first equals the number of intervals comfortably required for leg changeover, changeover is initiated. Changeover is left until the end of a planned movement to reduce unnecessary changeovers which are time consuming. For example, target points may change before changeover, requiring a new movement. Movement of the legs 6 and 8 during changeover follows the trapezoidal velocity profiles discussed previously with reference to Figure 3.

The robot 2 is initialised by `sav_init` in `sav.c`. This:

1. calls `sav_cleanup` which deletes remnants of old tasks and message queues from any previous runs and rearranges some task priorities.
2. calls `io_init` in `io.c` to initialise the drivers for the digital I/O modules 270.
3. calls `servo_init` in `servo.c` to initialise the LM628 addresses and parameters for the servo modules 274 and reads the current servo positions and implements closed loop position control to them.

- 26 -

4. calls `servo_power` in `servo.c` to prompt for an operator to provide power to the robot 2.
5. readies the robot for servo calibration. This checks the status of the
5 electromagnets 37 and solenoid valves 128 of both legs 6 and 8 to decide if they are up (free) or down (gripping). It then makes the minimum changes necessary to ensure that one leg is down and one up. This matches the robot 2 and software states with minimum risk of the robot 2 falling from, for example, a vertical surface.
10
6. calls `servo_calib` in `servo.c` to calibrate the servo motors 68, 70. This is achieved as follows:
 - 15 (a) Encoder counters for the controller 252 are zeroed and then all servo motors 68, 70 are slowly turned to drive the nuts 62, 74 towards the motors 68, 70 until the limit switches operate. These switches disable further drive of the motors 68, 70 in this direction, and the motors 68, 70 stop, but the position targets continue their slow change.
 - 20 (b) Each encoder counter is read when its limit switch actuates and when all have actuated, position commands to move each servo motor 68, 70 back to its switch position are issued.
 - 25 (c) Each servo is then commanded to move another 1.25 revolutions in this direction. When the next index pulse is observed from each encoder 256, its counter contents are saved, a predetermined offset added and a new move started to this position. When these moves are completed, the servo motors 68, 70 are all in their home position.
 - 30 (d) The encoder counters are then re-zeroed and the servo position commands reset to zero. This completes servo calibration and all subsequent servo positions are referenced to these home positions. Control returns to

- 27 -

sav_init.

7. The trajectory generator software is then initialised by **traj_init** in **traj.c**. This involves creation of mutex semaphores and initialisation of the homogeneous transform from the datum of the robot to the tool.
8. The servo task **servo_task** in **servo.c** and trajectory generator task **tragen_task** in **tragen.c** are then called and run synchronously, completing robot initialisation.
- Target points set using the control software in the specification include the desired velocity and the required position of the robot tool relative to its current position. A constant target point therefore causes continuous motion, provided the commanded velocity is non-zero. The function **traj_settar** in **traj.c**, sets target points. It can be called from a VxWorks shell by typing **traj_settar V, x, y**, where **V** is the desired velocity in m/s, **x** and **y** are the transverse and axial components of the position of the target point in millimetres. Control by a joystick is initiated by spawning the joystick task as listed in **joystick.c**. This reads the joystick and generates the arguments **V, x, y** and then calls **traj_settar**, as above. Control by the joystick simulates derivation of data from a feature sensor, possibly a mechanical probe or a light stripe system discussed for the first control system 200.

Following initialisation, the main loop, within **servo_task** in **servo.c** continues execution every servo interval awaiting a target point. It is the master which initiates execution of other functions and controls the robot 2. It interfaces with the trajectory generator **traj.c** via the task **tragen_task** in **tragen.c**. Communication between **servo.c** and **tragen.c** is via message queues which deliver two structures, **state** and **command**. These contain all the data to define the actual and desired state of the robot 2.

Within the main loop, the first operation is to compute the commanded velocity for each servo motor 68, 70 from the current and previous position commands and compare it with the maximum velocity. If it is within the limits, position and velocity commands are determined and the servo modules 274 instructed accordingly by

- 28 -

servo_load and initiated by **servo_move**. Next, the output signals for magnetic coils 37 and solenoids 128 of each leg are set as required from the **command** structure. This is done using **io_set**. As the **command** structure is invalid during the first pass, the variable **serial** which is false only on the first pass, inhibits erroneous setting of these outputs.

5

Next, the robot state is read to build the **state** structure. The servo positions are read into the **state** structure before **servo_load** above using **lm628_rdrp**. The state of each magnet and solenoid is read using **io_readback** and the **adhesion** element of the leg's state structure set true for each toe 10 if both its magnetic coil 37 and solenoid 128 are on.

10

The **state** structure is then sent to **tragen.c** using **msgQSend** and **servo.c** is suspended awaiting the next **command** message from **tragen.c**. Its suspension allows **tragen_task** (which has a lower priority) to run. When **tragen_task** receives the message using **msgQReceive**, it calls **traj.c**, passing pointers to the **state** and an empty **command** structure. When **traj.c** returns, the **command** structure is returned to **servo.c** using another message queue send/receive pair. Both **servo.c** and **tragen.c** have timeouts a little longer than the servo interval which halt execution via **sav_panic** if the messages are not passed in time. The purpose of using separate **servo_task** and **tragen_task** tasks with message queues is to isolate the code in **traj.c** from the mechanism of robot control. This improves the portability between VxWorks and Unix and also potentially allows recovery from failure of the code in **traj.c**.

15

20

After execution returns to **servo.c**, the status of the robot power and each servo motor 68, 70 is checked and if an anomaly is found, execution is halted via **sav_panic** and an appropriate message printed. If not, execution returns to the top of the main loop ready for the next servo interval.

25

The function **traj** in **traj.c** receives the robot **state** from **tragen_task** and calls it **svacl** internally. It computes the **command** structure, which it calls **svcom** internally, and returns it to **tragen_task**.

30

SUBSTITUTE SHEET

- 29 -

On the first pass (**serial** is **false**), **traj** initialises internal variables for the encoder values, status flags for each leg 6, 8, and homogeneous transforms for each leg 6, 8 relative to the robot datum. Functions for transform manipulation are defined in **math.c**. This data is kept in internal structures **leg0** and **leg1** and are accessed by the pointers **leg** and **leg_up**.

The flag **sts** is set to the value **NEWTAR** by **traj_settar** when a new target point has been accepted. If in addition, leg changeover is not occurring, it allows acquisition of a new target point by the function **get_newtar**. The function **traj_tar** is then called to compute servo position commands which **traj** places in the structure **svcom**.

The function **traj** next executes the logic controlling the operations in leg changeover, providing the velocity demand **vd** is non-zero. The changeover operations are controlled by the value of elements in the **leg** structure, **leg->no**, **leg->nm**, **leg->ns**, **leg->nu**, and **leg->nn**. The operation is executed whenever **seg_no** reaches an element value. The elements are computed by the function **traj_seg** from the number of servo interval time segments to be completed before the tool point reaches the current target point. Each of the controlling elements in the **leg** structure is computed from this value and the number of servo intervals required to execute the individual processes associated with leg changeover.

For example, **leg->no** is the number of trajectory segments remaining to be completed before the solenoid valves 128 of the leg 6 or 8 which is engaging the surface 18 must be actuated to drive the toes 10 down. When **seg_no** reaches **leg->no**, the solenoids 128 of both legs 6 and 8 are set to the value **SOLDOWN** which drives the legs down. At any time, the solenoids 128 of at least one leg 6 or 8 will be in this state, and their setting will have no effect. This practice is used in many places for both solenoids 128 and magnet coils 37 to improve the symmetry of the code.

When **seg_no** reaches **leg->nn**, leg changeover is complete, the status of each leg 6, 8 is exchanged and the pointers indicating the clamped leg, **leg** and lifted leg, **leg_up** are set. Control is then returned to **tragen_task**.

SUBSTITUTE SHEET

- 30 -

The function **traj_tar** computes an achievable target point and calls other functions to compute servo position commands which are passed to **traj.c**. At the beginning of each step **traj_tar** saves the datum transform of the target point to the clamped leg. The transform of each target point is computed with respect to the clamped leg 6 or 8. This is used by **traj_stck** to check if the target is within the mobility region of the robot. If not, a new target point in the vehicle coordinate system midway between the tool point and the original target point is computed and retested. Closer target points continue to be computed until one within the mobility region is obtained. It is then returned to **traj_tar**. The function **traj_seg** is then called and the number of servo interval segments required to move the tool point to the target is computed (the function **inv_kin** is defined in **kine.c**). If the result is non-positive, leg changeover is initiated. In either case, **traj_step** is called.

The function **traj_step** computes servo positions to reach the target point. It calls **traj_gtst** to compute the position and angle of the shaft 22 of the clamped leg 6 or 8. It then calls **traj_axis** for each leg. This calculates the servo positions of the clamped leg to achieve this position and angle and the servo positions of the other leg as appropriate. **Traj_step** also updates the number of servo interval segments done within this step. If the result indicates that the target has been reached, the flag **sts1** is set to the value **NEWSTEP** and the transform of the target point converted to be referenced to the newly clamped leg.

The function **traj_axis** computes servo positions for each leg 6, 8. It has arguments which signify the mode of a leg. If the mode is **LEGDOWN** the leg is clamped to the surface and the function **traj_down** is called. This calls **inv_kin** in **kine.c** to compute the position and angle of shaft 22, then **leg_axis** to convert to ball screw positions, and **axis_enc** to compute servo positions in encoder counts.

If the mode is **LEGUP**, the leg is up and if **seg_no** indicates that the leg does not yet have to accelerate to catch up to the clamped leg, **traj_reset** is called. This implements a trapezoidal velocity profile to return the leg to its home position.

- 31 -

If the mode is **LEGUP**, and **seg_no** indicates that the leg does have to accelerate to catch up to the clamped leg, **traj_catch** is called. This ramps the leg velocity up to that of the clamped leg. If the leg is in the overlapping phase (both legs clamped), **traj_overlap** is called as the velocities of each leg are matched.

5

Lastly, **col_clsn** is called to check for collisions before the servo positions are passed back to **tragen.c** for execution.

The code in **collision.c** represents the rollers 32, legs 6, 8, shafts 22, and toes 10 of the robot 2 as simple lines and circles. Each is defined relative to the positions and angles of the shafts 22 of each leg 6, 8. The code checks for collisions between:

- (a) the circles representing toes 10,
- 15 (b) between these circles and those representing the rollers 32,
- (c) between the circle representing the shaft 22 of leg 6 and the lines representing the struts 20 of leg 8.

20 If a potential collision is detected, the function **sav_panic** is called to stop the robot 2 and an appropriate message displayed.

- 32 -

CONTROL SOFTWARE

collision.c

```

5  #ifdef VXWORKS
   #include <stdioLib.h>
   #else
   #include <stdio.h>
   #endif
10 #include <math.h>
   #include "traj.h"

   ELEMENT    *c0[4], *c1[4], *c2[4], *l1[3], *l2[5];
   double      ro10[2], ro11[2], ro12[2], ro13[2], ro20[2],
15             ro21[2], ro22[2], ro23[2], ro24[2], ro25[2];

   col_init()
   {
20   int  i;
       for(i=0; i<=3; i++){
           c0[i] = (ELEMENT *)malloc(sizeof(ELEMENT));
           c1[i] = (ELEMENT *)malloc(sizeof(ELEMENT));
           c2[i] = (ELEMENT *)malloc(sizeof(ELEMENT));
25       }
       for(i=0; i<=2; i++){
           l1[i] = (ELEMENT *)malloc(sizeof(ELEMENT));
       }
       for(i=0; i<=4; i++){
30         l2[i] = (ELEMENT *)malloc(sizeof(ELEMENT));
       }

       for(i = 0; i <=3; i++){
           c0[i]->px = (double *)malloc(sizeof(double));
35         c0[i]->py = (double *)malloc(sizeof(double));
           c1[i]->px = (double *)malloc(sizeof(double));
           c1[i]->py = (double *)malloc(sizeof(double));
           c2[i]->px = (double *)malloc(sizeof(double));
           c2[i]->py = (double *)malloc(sizeof(double));
40       }
       for(i = 0; i<= 2; i++){
           l1[i]->px = (double *)malloc(2 * sizeof(double));
           l1[i]->py = (double *)malloc(2 * sizeof(double));
       }
45       for(i = 0; i<= 4; i++){
           l2[i]->px = (double *)malloc(2 * sizeof(double));
           l2[i]->py = (double *)malloc(2 * sizeof(double));
       }
       /* initialize each reference point */
50       /* wheels of the main frame */
       c0[0]->px[0] = R00x;
       c0[0]->py[0] = R00y;
       c0[1]->px[0] = R01x;
55       c0[1]->py[0] = R01y;

```

SUBSTITUTE SHEET

- 33 -

```

c0[2]->px[0] = R02x;
c0[2]->py[0] = R02y;
c0[3]->px[0] = R03x;
c0[3]->py[0] = R03y;
5
    /* Set constants */
    for(i=0; i<=3; i++)
        c0[i]->s = SC2;

10    for(i=1; i<=3; i++){
        c1[i]->s = SC1;
        c2[i]->s = SC1;
    }
    for(i=0; i<=2; i++){
15        l1[i]->s = SL0;
        l2[i]->s = SL0;
    }
    c1[0]->s = c2[0]->s = SL0;
    l1[0]->d = l1[1]->d = D10;
20    l1[2]->d = l2[2]->d = D12;
    l2[0]->d = l2[1]->d = D20;
    l2[3]->d = l2[4]->d = D23;
    l2[3]->s = l2[4]->s = SL0;

25    /* initial reference points for leg1*/
    ro10[0]= R10x;
    ro10[1]= R10y;
    ro11[0] = R11x;
30    ro11[1]= R11y;
    ro12[0]= R12x;
    ro12[1]= R12y;
    ro13[0]= R13x;
    ro13[1]= R13y;
35    /* initial reference points for leg2*/
    ro20[0] = R20x;
    ro20[1] = R20y;
    ro21[0] = R21x;
40    ro21[1] = R21y;
    ro22[0] = R22x;
    ro22[1] = R22y;
    ro23[0] = R23x;
    ro23[1] = R23y;
45    ro24[0] = R24x;
    ro24[1] = R24y;
    ro25[0] = R25x;
    ro25[1] = R25y;
50 }

col_trn(x1, y1, x2, y2, a11, a12)
55 double x1, y1, x2, y2, a11, a12;
{

```

SUBSTITUTE SHEET

- 34 -

```

double    r10[2], r11[2], r12[2], r13[2], r20[2],
          r21[2], r22[2], r23[2], r24[2], r25[2];

double    t1[3][3], t2[3][3];
5      trans_mat(x1, y1, a11, t1);
      trans_mat(x2, y2, a12, t2);
      transf(t1, r10, ro10);
      transf(t1, r11, ro11);
      transf(t1, r12, ro12);
10     transf(t1, r13, ro13);
      transf(t2, r20, ro20);
      transf(t2, r21, ro21);
      transf(t2, r22, ro22);
      transf(t2, r23, ro23);
15     transf(t2, r24, ro24);
      transf(t2, r25, ro25);

20     /* Toes, shaft and leg frame of leg1 */
      l1[0]->px[0] = l1[1]->px[0] = l1[2]->px[0] =
      c1[0]->px[0] = r10[0];
      l1[0]->py[0] = l1[1]->py[0] = l1[2]->py[0] =
      c1[0]->py[0] = r10[1];
25     l1[0]->px[1] = c1[1]->px[0] = r11[0];
      l1[0]->py[1] = c1[1]->py[0] = r11[1];
      l1[1]->px[1] = c1[2]->px[0] = r12[0];
      l1[1]->py[1] = c1[2]->py[0] = r12[1];
      l1[2]->px[1] = c1[3]->px[0] = r13[0];
30     l1[2]->py[1] = c1[3]->py[0] = r13[1];

      /* Toes, shaft and leg frame of leg2 */
      l2[0]->px[0] = l2[1]->px[0] = l2[2]->px[0] =
      c2[0]->px[0] = r20[0];
35     l2[0]->py[0] = l2[1]->py[0] = l2[2]->py[0] =
      c2[0]->py[0] = r20[1];
      l2[0]->px[1] = l2[3]->px[1] = r24[0];
      l2[0]->py[1] = l2[3]->py[1] = r24[1];
      l2[1]->px[1] = l2[4]->px[1] = r25[0];
40     l2[1]->py[1] = l2[4]->py[1] = r25[1];
      l2[2]->px[1] = c2[2]->px[0] = r23[0];
      l2[2]->py[1] = c2[2]->py[0] = r23[1];
      l2[3]->px[0] = c2[1]->px[0] = r21[0];
      l2[3]->py[0] = c2[1]->py[0] = r21[1];
45     l2[4]->px[0] = c2[3]->px[0] = r22[0];
      l2[4]->py[0] = c2[3]->py[0] = r22[1];

}

50 col_line_circle(1, c)
   ELEMENT *c, *l;
   {
   double a, b, r, th, th1, th2, A, B, C;

55     B = r = sqrt((c->px[0]-l->px[0]) * (c->px[0]-l->px[0])
      + (c->py[0]-l->py[0]) * (c->py[0]-l->py[0]));

```

- 35 -

```

    A = sqrt((c->px[0]-l->px[1]) * (c->px[0]-l->px[1])
      + (c->py[0]-l->py[1]) * (c->py[0]-l->py[1]));
    C = sqrt((l->px[1]-l->px[0]) * (l->px[1]-l->px[0])
      + (l->py[1]-l->py[0]) * (l->py[1]-l->py[0]));
5
    th = acos((B * B + C * C - A * A)/(2.0 * B * C));
    if(th > Pion2)
        return 0;

10
    a = r * sin(th);
    b = r * cos(th);
    if((a <= c->s + l->s + MARG) && b <= l->d)
    {
15    printf("a = %f, b=%f, l->s=%f\n", a, b);
        return 1;
    }
    else
        return 0;
20 }

col_circle_circle(c1, c2)
ELEMENT *c1, *c2;
25 {
    double r;
    r = sqrt((c1->px[0]-c2->px[0]) * (c1->px[0]-c2->px[0])
      + (c1->py[0]-c2->py[0]) * (c1->py[0]-c2->py[0]));
    if(r<=c1->s + c2->s + MARG)
30        return 1;
    else
        return 0;
}

35

col_cllsn(l1_y1, l1_y2, l2_y1, l2_y2)
double l1_y1, l1_y2, l2_y1, l2_y2;
{
40 int co_flag = 0;
    double al1, al2;

    al1 = atan2(l1_y2 - l1_y1, ARMLEN);
    al2 = atan2(l2_y1 - l2_y2, ARMLEN);
45
    col_trn(25., l1_y1, -25., l2_y1, al1, al2);

    /* Line \ Circle collision */

50    /*leg1 circles \ leg2 line */
    if(col_line_circle(l2[0], c1[0])){
        sav_panic("l20 collides with c10\n");
        co_flag = 1;
    }
55    if(col_line_circle(l2[1], c1[0])){
        sav_panic("l21 collides with c10\n");
    }

```

- 36 -

```

        co_flag = 1;
    }
    if(col_line_circle(l2[3], c1[0])){
        sav_panic("l23 collides with c10\n");
5       co_flag = 1;
    }
    if(col_line_circle(l2[4], c1[0])){
        sav_panic("l24 collides with c10\n");
10      co_flag = 1;
    }

    if(col_line_circle(l2[0], c1[1])){
        sav_panic("l20 collides with c11\n");
15      co_flag = 1;
    }
    if(col_line_circle(l2[3], c1[1])){
        sav_panic("l23 collides with c11\n");
        co_flag = 1;
    }
20    if(col_line_circle(l2[2], c1[1])){
        sav_panic("l22 collides with c11\n");
        co_flag = 1;
    }

25
    if(col_line_circle(l2[1], c1[2])){
        sav_panic("l21 collides with c12\n");
        co_flag = 1;
    }
30    if(col_line_circle(l2[4], c1[2])){
        sav_panic("l24 collides with c12\n");
        co_flag = 1;
    }
    if(col_line_circle(l2[2], c1[2])){
35      sav_panic("l22 collides with c12\n");
        co_flag = 1;
    }

    if(col_line_circle(l2[0], c1[3])){
40      sav_panic("l20 collides with c13\n");
        co_flag = 1;
    }
    if(col_line_circle(l2[1], c1[3])){
        sav_panic("l21 collides with c13\n");
45      co_flag = 1;
    }
    if(col_line_circle(l2[3], c1[3])){
        sav_panic("l23 collides with c13\n");
50      co_flag = 1;
    }
    if(col_line_circle(l2[4], c1[3])){
        sav_panic("l24 collides with c13\n");
55      co_flag = 1;
    }

    /*leg2 circles \ leg1 line */

```

SUBSTITUTE SHEET

- 37 -

```

    if(col_line_circle(l1[0], c2[1])){
        sav_panic("l10 collides with c21\n");
        co_flag = 1;
    }
5   if(col_line_circle(l1[2], c2[1])){
        sav_panic("l12 collides with c21\n");
        co_flag = 1;
    }

10  if(col_line_circle(l1[0], c2[2])){
        sav_panic("l10 collides with c22\n");
        co_flag = 1;
    }
    if(col_line_circle(l1[1], c2[2])){
15  sav_panic("l11 collides with c22\n");
        co_flag = 1;
    }

    if(col_line_circle(l1[1], c2[3])){
20  sav_panic("l11 collides with c23\n");
        co_flag = 1;
    }
    if(col_line_circle(l1[2], c2[3])){
25  sav_panic("l12 collides with c23\n");
        co_flag = 1;
    }

    /* Circle \ circle collision */
    /* Between leg1 and leg2 */
30
    if(col_circle_circle(c1[1], c2[1])){
        sav_panic("c11 collides with c21\n");
        co_flag = 1;
    }
35  if(col_circle_circle(c1[1], c2[2])){
        sav_panic("c11 collides with c22\n");
        co_flag = 1;
    }
    if(col_circle_circle(c1[2], c2[2])){
40  sav_panic("c12 collides with c22\n");
        co_flag = 1;
    }
    if(col_circle_circle(c1[2], c2[3])){
        sav_panic("c12 collides with c23\n");
45  co_flag = 1;
    }
    if(col_circle_circle(c1[3], c2[3])){
        sav_panic("c13 collides with c23\n");
        co_flag = 1;
50  }
    if(col_circle_circle(c1[3], c2[1])){
        sav_panic("c13 collides with c21\n");
        co_flag = 1;
    }
55
    /* Between legs and rollers */

```

SUBSTITUTE SHEET

- 38 -

```

        if(col_circle_circle(c1[3], c0[2])){
            sav_panic("c13 collides with c02\n");
            co_flag = 1;
        }
5      if(col_circle_circle(c2[3], c0[2])){
            sav_panic("c23 collides with c02\n");
            co_flag = 1;
        }
10     if(col_circle_circle(c1[2], c0[1])){
            sav_panic("c12 collides with c01\n");
            co_flag = 1;
        }
        if(col_circle_circle(c2[2], c0[1])){
15         sav_panic("c22 collides with c01\n");
            co_flag = 1;
        }
        return co_flag;
    }

20  /*
    * joystick.c
    */
    #include <vxWorks.h>
    #include <taskLib.h>
25  #include <ioLib.h>
    #include <semLib.h>

    #include "io.h"
    #include "joystick.h"
30

    #define JOFF_X 44
    #define JOFF_Y 41

35  #define JDEAD 1

    #define JX_MAX      18.0      /* joystick max. value in +x
    direction */
    #define JXM_MAX     -24.0     /* joystick max. value in -x
40  direction */
    #define JY_MAX      20.0      /* joystick max. value in y
    direction */
    #define X_MAX       10.0      /* max. setpoint value in x
    direction */
45  #define Y_MAX       75.0      /* max. setpoint value in y
    direction */
    #define Y_MIN       40.0      /* min. setpoint value in y
    direction */
    #define V_MAX       0.02      /* max. speed (m/s) */
50  #define V_MIN       0.01      /* speed at which the sav is
    turning max.

                                * curvature */
    #define XINC_MAX    25        /* maximum joystick
    positional increment in

55  * each set point */
    #define YINC_MAX    25        /* maximum joystick

```

SUBSTITUTE SHEET

- 39 -

```

velocity increment for
                                * each set point */
#define      sgn(x)      (x >= 0 ? 1 : -1)

5  int      joystick_debug = 0;
   int      jxp = 0, jyp = 0;

   int      joystick_read(int *x, int *y)
       /*
10      * Reads a signed X and Y value from the joystick ADCs
        and returns into x
        * and y pointers. Returns square of absolute joystick
        offset
        */
15  {
       int      jx, jy;

       /* take X reading */
       io_set(IO_JOYSEL, 1);
20      jx = (int) (
               (io_get(IO_JOY0) << 0) |
               (io_get(IO_JOY1) << 1) |
               (io_get(IO_JOY2) << 2) |
               (io_get(IO_JOY3) << 3) |
25      (io_get(IO_JOY4) << 4) |
               (io_get(IO_JOY5) << 5)
               );

       /* take Y reading */
30      io_set(IO_JOYSEL, 0);
       jy = (int) (
               (io_get(IO_JOY0) << 0) |
               (io_get(IO_JOY1) << 1) |
               (io_get(IO_JOY2) << 2) |
35      (io_get(IO_JOY3) << 3) |
               (io_get(IO_JOY4) << 4) |
               (io_get(IO_JOY5) << 5)
               );

40      if (joystick_debug)
          printf("%2d %2d ", jx, jy);

       /* elide offset count */
       jx -= JOFF_X;
45      jy -= JOFF_Y;

       if (joystick_debug)
          printf("%2d %2d ", jx, jy);

50      /* implement deadband of size JDEAD */
       jx = (jx >= JDEAD) ? jx - JDEAD : ((jx <= -JDEAD) ? jx
+ JDEAD : jx);
       jy = (jy >= JDEAD) ? jy - JDEAD : ((jy <= -JDEAD) ? jy
+ JDEAD : jy);
55      if (joystick_debug)

```

SUBSTITUTE SHEET

- 40 -

```

        printf("%2d %2d\n", jx, jy);

        /* return result */
        if (x) *x = jx;
5       if (y) *y = jy;

        return jx * jx + jy * jy;
    }

10    int joystick_show(void)
    {
        int x, y;

15        joystick_read(&x, &y);
        printf("%d %d\n", x, y);
        return 0;
    }

20    static joystick_tar(int jx, int jy)
    {
        double x, y, v;

25        if (jx >= 0) {
            x = (double) jx * (X_MAX/JX_MAX);
            y = -x * (Y_MAX - Y_MIN)/X_MAX + Y_MAX;
        } else {
            x = -(double)jx * (X_MAX/JXM_MAX);
30            y = x * (Y_MAX - Y_MIN)/X_MAX + Y_MAX;
        }

        if ((double) jy < 0.0)
            (double) jy = 0.0;
35        if (jx >= 0)
            v = (double)jy/JY_MAX * (V_MAX - (double)jx/JX_MAX *
(V_MAX - V_MIN));
        else
            v = (double)jy/JY_MAX * (V_MAX - (double)jx/JXM_MAX *
40 (V_MAX - V_MIN));

        traj_settar(v, x, y);
    }

45    joystick_task(void)
    {
        int jx, jy;
50        int djx, djy; /* delta x and y */

        for (;;) {
            joystick_read(&jx, &jy);
            djx = jx - jxp;
55            djy = jy - jyp;

```

- 41 -

```

        if (djx != 0 || djy != 0 || jy == 0) {
            /* send a new set point when there is a change on
            the joystick or
            * the joystick is at the centre (in this way the
5 robot will stop
            * when the handle is released, otherwise if it is
            released
            * during the changeover phase in which any new
            set point is
10 * ignored, the robot will keep moving at the
            speed set before
            * the changeover
            */

15         if (fabs(djx) >= XINC_MAX)
            djx = XINC_MAX * sgn(djx);
            if (fabs(djy) >= YINC_MAX)
            djy = YINC_MAX * sgn(djy);
            jxp = jxp + djx;
20         jyp = jyp + djy;
            joystick_tar(jxp, jyp);
        }
        taskDelay(20);
    }
25 }

/*
 * servo.c -- servo system functions
 */
30 #include <vxWorks.h>
#include <taskLib.h>
#include <msgQLib.h>
#include <stdioLib.h>
#include <ssclib.h>
35 #include <vme.h>

#include "sav.h"
#include "vipc610.h"
#include "io.h"
40 #include "servo.h"
#include "lm628.h"

static struct {
45     /*
        * Information table for the servo chips, including
        position loop
        * gain parameters, 10 channels of limit switches,
        lines per encoder
50     * and calibration offsets
        */

        /* chip addresses (bus and local) */
        void *base;
55     volatile byte *addr;

```

SUBSTITUTE SHEET

- 42 -

```

/* servo chip parameters */
byte ds;
word kp, ki, kd, il;

5  /* limit switch channels */
   int cw, ccw, inhibit;

/* encoder/speed values */
unsigned lines, maxrpm, maxvel;

10 /* calibration index offset */
    int offs;

} servo[NSERVO] = {
15  {
    VIPC610_IPC + 1, 0,
    0, 77, 8, 192, 100,
    IO_CW0, IO_CCW0, IO_INH0,
    500, 6000, 0,
20    10146
  }, {
    VIPC610_IPC + 5, 0,
    0, 49, 5, 123, 100,
    IO_CW1, IO_CCW1, IO_INH1,
25    500, 4500, 0,
    1810
  }, {
    VIPC610_IPD + 1, 0,
    0, 49, 5, 123, 100,
30    IO_CW2, IO_CCW2, IO_INH2,
    500, 4500, 0,
    1997
  }, {
    VIPC610_IPD + 5, 0,
35    0, 77, 8, 192, 100,
    IO_CW3, IO_CCW3, IO_INH3,
    500, 6000, 0,
    10497
  }
40 };

servo_move(int check)
/*
45  * Executes a move previously set up by servo_load()
  * If check is true, checks power supply status
  */
{
  int channel;

50  /* check we still have servo power */
  if (check && io_get(IO_PM1) == 0)
    sav_panic("servo_move: +64VDC umbilical power
55  failed\n");

  for (channel = 0; channel < NSERVO; ++channel)

```

SUBSTITUTE SHEET

- 43 -

```

        lm628_stt(servo[channel].addr);
        return 0;
    }

5   servo_wait(int check)
    /*
     * Polls the completion of a move started with
     servo move
10   * If check is true, checks power supply status
     */
    {
        int channel;
        byte done = 0;
15   while (! (done & LM628_S_TDONE)) {
            taskDelay(1);
            done = 0xff;
            for (channel = 0; channel < NSERVO; ++channel)
20             done &= lm628_rdstat(servo[channel].addr);

            /* check we still have servo power */
            if (check && io_get(IO_PM1) == 0)
                sav_panic("servo_wait: +64VDC umbilical power
25 failed\n");
        }
        return 0;
    }

30   int servo_show(void)
    /*
     * Prints a status line for all the servos
     */
35   {
        int channel = 0;

        lm628_show(NULL);
        for (channel = 0; channel < NSERVO; ++channel)
40         lm628_show(servo[channel].addr);
        return 0;
    }

45   int servo_read(int pos[])
    /*
     * Reads current position of all servos into pos
     */
    {
50     int channel;
        for (channel = 0; channel < NSERVO; ++channel)
            lm628_rdrp(servo[channel].addr, pos + channel);
        return 0;
    }

55

```

SUBSTITUTE SHEET

- 44 -

```

servo_load(int pos[], unsigned v[], unsigned a[])
/*
 * Load trajectory, accepting vectors of position,
velocity
5      * and acceleration in units of encoder counts,
counts/sec and
      * counts/sec/sec. a and/or v may be NULL, indicating
carry-over
      * of previous values respectively
10     */
{
    int channel;
    word flags = LM628_T_PL;
    unsigned vel[NSERVO], acc[NSERVO];
15
    /* normalise velocity to counts/sec, not counts/int */
    if (v) {
        flags |= LM628_T_VL;
        for (channel = 0; channel < NSERVO; ++channel)
20            vel[channel] = (0x10000 * v[channel])/SRATE;
    }

    /* normalise acceleration to counts/sec/sec, not
counts/int/int */
25    if (a) {
        flags |= LM628_T_AL;
        for (channel = 0; channel < NSERVO; ++channel) {
            acc[channel] = ((0x10000 *
a[channel])/SRATE)/SRATE;
30
            /* the chip manual says to do this ... */
            lm628_ltrj(servo[channel].addr,
(word)LM628_T_MOFF);
        }
35    }

    /* load the servo traj buffer */
    for (channel = 0; channel < NSERVO; ++channel)
        if ((flags & LM628_T_VL) && (flags & LM628_T_AL))
40            lm628_ltrj(
                servo[channel].addr,
                (word)(LM628_T_PL | LM628_T_VL | LM628_T_AL),
                acc[channel],
                vel[channel],
45                pos[channel]
            );
        else
            if (flags & LM628_T_VL)
                lm628_ltrj(
50                    servo[channel].addr,
                    (word)(LM628_T_PL | LM628_T_VL),
                    vel[channel],
                    pos[channel]
                );
            else
55                if (flags & LM628_T_AL)

```

SUBSTITUTE SHEET

- 45 -

```

        lm628_ltrj(
            servo[channel].addr,
            (word)(LM628_T_PL | LM628_T_AL),
            acc[channel],
5            pos[channel]
        );
    else
        lm628_ltrj(
            servo[channel].addr,
10            (word)(LM628_T_PL),
            pos[channel]
        );
    return 0;
}
15

int servo_init(int mode)
/*
20  * Initialise the servo system
  *
  * The local addresses of the servo chips and some
  other stuff, stored in
  * the servo array, is initialised. If it is a cold
25 start (determined by
  * the mode parameter) the servos are reset and loaded
  with the position
  * loop compensator values from the servo[] array. The
  servo positions are
30  * then read and a move to this position initiated.
  This effectively start
  * the servo position loop.
  */
{
35  int channel, pos[NSERVO];
  unsigned vel[NSERVO], acc[NSERVO];

  /* set up servo array */
  for (channel = 0; channel < NSERVO; ++channel) {
40      /* get local address of LM628 servo chip */
      sysBusToLocalAdrs(
          VME_AM_USR_SHORT_IO,
          servo[channel].base,
45          &(servo[channel].addr)
      );

      /* calculate the maximum velocity entry (lines/s) */
      servo[channel].maxvel =
50      (servo[channel].maxrpm * servo[channel].lines *
        4)/60;
  }

  /* reset the chip parameters if cold start */
55  if (mode == SERVO_COLD)
      for (channel = 0; channel < NSERVO; ++channel) {

```

SUBSTITUTE SHEET

- 46 -

```

    /*
    * reset chips, set 12-bit port, clear interrupts,
    * set breakpoint way off range
    */
5      lm628_reset(servo[channel].addr);
      lm628_port12(servo[channel].addr);
      lm628_sbpa(servo[channel].addr, 0x3fffffff);
      lm628_rsti(servo[channel].addr, 0xff);

10     /*
    * Set position loop gains (it is recommended to
    re-tune both
    * position and velocity servos after the whole
    mechanism is
15     * assembled)
    */
      lm628_lfil(
          servo[channel].addr,
          servo[channel].ds,
20         servo[channel].kp,
          servo[channel].ki,
          servo[channel].kd,
          servo[channel].il
      );
25     lm628_udf(servo[channel].addr);
    }

    /* the velocity and acceleration don't matter much */
    for (channel = 0; channel < NSERVO; ++channel) {
30         vel[channel] = servo[channel].lines * 4;
         acc[channel] = servo[channel].lines * 4;
    }

    /* in any case, read current servo position and servo
35 to it */
    servo_read(pos);
    servo_load(pos, vel, acc);
    servo_move(0);
    servo_wait(0);
40
    return 0;
}

45 int servo_power(int power)
    /*
    * controls servo and magnet power (0 means off,
    otherwise on)
    */
50 {
    /* switch off case */
    if (power == 0) {
        io_set(IO_PSI, 0);
        return 0;
55    }
}

```

- 47 -

```

    printf("Turn walker power on ... ");
    io_set(IO_PSI, 1);

    /* poll the human */
5   while (! (io_get(IO_PM1) && io_get(IO_PM2)))
        taskDelay(1);

    /* it's on now */
    printf("taa\n");
10   taskDelay(25);
    return 0;
}

15 int servo_calib(void)
    /*
        * The general idea is to move the two legs (with one
        potentially clamped
        * to the surface) to the zero position, and to zero
20 the servo position
        * counters.
        */
    {
        int channel, ndone = 0;
25     int pos[NSERVO], index[NSERVO], done[NSERVO];
        unsigned vel[NSERVO], acc[NSERVO];

        /*
            * The first stage is to zero the counters, as it is
30 assumed that they
            * contain nothing useful. All servos are then moved
            slowly until the
            * limit switches operate. This stops motion by
            disabling the velocity
35     * servo, but the position trajectory generator in the
            LM628 ploughs
            * on regardless. This is why we disable the LM628
            breakpoint and position
            * error interrupt stuff first.
40     */

        /* zero all the position counters */
        for (channel = 0; channel < NSERVO; ++channel)
            lm628_dfh(servo[channel].addr);
45

        /* now, move all at lrev/s toward the end of travel */
        for (channel = 0; channel < NSERVO; ++channel) {
            pos[channel] = -1000000;
            vel[channel] = servo[channel].lines * 4;
50     acc[channel] = servo[channel].lines * 4;
            done[channel] = 0;
        }

        /* load servo and start the motion */
55     servo_load(pos, vel, acc);
        servo_move(1);

```

SUBSTITUTE SHEET

- 48 -

```

/*
 * Here we poll the limit switches until all of the
servos have reached
 * them. The first time each does, we read its current
5 encoder position
 * and start a new move back to that position.
 */

/* poll the limit switches until they all arrive */
10 for (ndone = 0; ndone < NSERVO; ) {

    for (channel = 0; channel < NSERVO; ++channel) {
        if (io_get(servo[channel].cw) == 0 &&
15 !done[channel]) {

            /* read current servo position and move to it */
            lm628_rdrp(servo[channel].addr, pos + channel);

            lm628_ltrj(servo[channel].addr, (word)LM628_T_PL,
20 pos[channel]);
            lm628_stt(servo[channel].addr);
            ++done[channel];
            ++ndone;
        }
25     }
    taskDelay(1);
}
servo_wait(1);

30

/*
 * Now we prime the servo to catch the next index pulse
and start a
 * move of at least one revolution (5 times the lines
35 per revolution).
 */
for (channel = 0; channel < NSERVO; ++channel) {
    pos[channel] += servo[channel].lines * 5;

40     /* clear any previous index status */
    lm628_rsti(servo[channel].addr, (word)LM628_I_INDEX);
    lm628_sip(servo[channel].addr);
    done[channel] = 0;
}
45 servo_load(pos, NULL, NULL);
servo_move(1);

/*
 * When the chip indicates it has observed an index, we
50 load it into
 * index[], and compute a new target position pos[] by
adding the
 * calibration offset to index[], and start a new move
to there
55 * At the end of it all we should be at the "home"
position

```

SUBSTITUTE SHEET

- 49 -

```

    */
    for (ndone = 0; ndone < NSERVO; ) {
5      for (channel = 0; channel < NSERVO; ++channel) {
          if ((lm628_rdstat(servo[channel].addr) &
LM628_S_INDEX)
          && !done[channel]
          ) {
10         /* read servo index position */
            lm628_rdip(servo[channel].addr, index + channel);

            /* compute position of home for this axis */
            pos[channel] = index[channel] +
15     servo[channel].offs;

            /* and start a move there */
            lm628_ltrj(servo[channel].addr, (word)LM628_T_PL,
pos[channel]);
20         lm628_stt(servo[channel].addr);

            ++done[channel];
            ++ndone;
        }
25     }
        taskDelay(1);
    }
    servo_wait(1);

30     /* finally zero the servos again */
    for (channel = 0; channel < NSERVO; ++channel) {
        lm628_dfh(servo[channel].addr);
        pos[channel] = 0;
35     }

    /* and do a move to zero */
    servo_load(pos, NULL, NULL);
    servo_move(1);
40     servo_wait(1);

    /* we are now at the "home" position */
    return 0;
45 }

int servo_task(unsigned long period)
/*
    * mode is the startup mode; period is the tragen rate
50 in us
    *
    * This is the centre of the motion system. It loops
    periodically, feeds
    * and waters the servo chips, and sends and receives
55 position status/
    * command messages to/from the trajectory generator

```

SUBSTITUTE SHEET

- 50 -

```

task.
    *
    * The servo task is started by sav_init()
    */
5  {
    SSC_ID clock;
    savmsg_servo_tragen state;
    savmsg_tragen_servo command;
    unsigned long serial = 0;
10   unsigned base, timeout = WAIT_FOREVER;
    int current[NSERVO], channel;

    /* check we still have servo power */
15   if (io_get(IO_PM1) == 0)
        sav_panic("servo_task: +64VDC umbilical power
failed\n");

    /* obtain a timing semaphore */
20   sscBaseGet(&base);
    clock = sscCreate(period/base, 0);
    period = ((int)(period/base)) * base;

    /* read initial position */
    servo_read(current);

    /* set the servos for high acceleration */
    for (channel = 0; channel < NSERVO; ++channel) {
30         /* the chip manual says to do this ... */
        lm628_ltrj(servo[channel].addr, (word)LM628_T_MOFF);
        lm628_ltrj(servo[channel].addr, (word)LM628_T_AL,
0x3fffffff);
35         lm628_stt(servo[channel].addr);

        /* copy current position to command, so that the first
pass */
        /* through the loop generates sensible velocity (i.e.
40 zero) */
        command.position[channel] = current[channel];
    }
    servo_wait(1);

    /* main loop */
45   for (sscEnable(clock); sscWait(clock) == OK; ++serial)
    {
        unsigned velocity[NSERVO];

50         /*
        * We deal with the command issued on the previous
iteration
        * of the servo loop. This is done at the top of the
loop so that
55         * the timing is accurate. The chips have to be
programmed with

```

SUBSTITUTE SHEET

- 51 -

```

    * both the command position and the velocity required
so they
    * get to that command in exactly one servo loop
period
5    */

    /* for each servo chip */
    for (channel = 0; channel < NSERVO; ++channel) {

10        /* compute velocity for this segment (in
counts/sec) */
        velocity[channel] = (abs(
            command.position[channel] - current[channel]
        ) * 1000000)/period;

15        /* compare against velocity limits */
        if (velocity[channel] > servo[channel].maxvel)
            sav_panic(
20 limit %u\n",
                "servo_task: servo %d: serial %d: velocity
                channel, serial, velocity[channel]
            );

        /* copy command into current, ready for next time
25 */
        current[channel] = command.position[channel];

        /* read the actual position into the state
structure */
30 lm628_rdrp(servo[channel].addr, state.position +
channel);
    }

    /* load servo chips and start the move segment */
35 servo_load(command.position, velocity, NULL);
    servo_move(1);

    /*
    * Now we execute the remainder of the command
40 message, for
    * the magnet drive and the leg solenoids. These are
specified
    * in the leg[] structure
    */
45 if (serial) {
        io_set(IO_MEN0, command.leg[0].magnet ? MAG_ON :
MAG_OFF);
        io_set(IO_SOL0A, command.leg[0].solenoid_a ?
SOL_DOWN : SOL_UP);
50 io_set(IO_SOL0B, command.leg[0].solenoid_b ?
SOL_DOWN : SOL_UP);
        io_set(IO_SOL0C, command.leg[0].solenoid_c ?
SOL_DOWN : SOL_UP);

55 io_set(IO_MEN1, command.leg[1].magnet ? MAG_ON :
MAG_OFF);

```

SUBSTITUTE SHEET

- 52 -

```

        io_set(IO_SOL1A, command.leg[1].solenoid_a ?
SOL_DOWN : SOL_UP);
        io_set(IO_SOL1B, command.leg[1].solenoid_b ?
SOL_DOWN : SOL_UP);
5      io_set(IO_SOL1C, command.leg[1].solenoid_c ?
SOL_DOWN : SOL_UP);
    }

10    /*
        * Now we build up the message to send to the tragen
        task. This
        * contains the current motion system state and
        standing commands.
15    * Note that the position is already there from the
        above.
        */

        state.serial = serial;
20      state.period = period;

        /* leg and magnet state */
        state.leg[0].magnet = (io_readback(IO_MEN0) == MAG_ON)
? 1 : 0;
25      state.leg[0].solenoid_a = (io_readback(IO_SOLA) ==
SOL_DOWN) ? 1 : 0;
        state.leg[0].solenoid_b = (io_readback(IO_SOLB) ==
SOL_DOWN) ? 1 : 0;
        state.leg[0].solenoid_c = (io_readback(IO_SOLC) ==
30 SOL_DOWN) ? 1 : 0;

        state.leg[0].adhesion_a= state.leg[0].solenoid_a &&
state.leg[0].magnet;
        state.leg[0].adhesion_b= state.leg[0].solenoid_b &&
35 state.leg[0].magnet;
        state.leg[0].adhesion_c= state.leg[0].solenoid_c &&
state.leg[0].magnet;

        state.leg[1].magnet = (io_readback(IO_MEN1) == MAG_ON)
40 ? 1 : 0;
        state.leg[1].solenoid_a = (io_readback(IO_SOL1A) ==
SOL_DOWN) ? 1 : 0;
        state.leg[1].solenoid_b = (io_readback(IO_SOL1B) ==
SOL_DOWN) ? 1 : 0;
45      state.leg[1].solenoid_c = (io_readback(IO_SOL1C) ==
SOL_DOWN) ? 1 : 0;

        state.leg[1].adhesion_a= state.leg[1].solenoid_a &&
state.leg[1].magnet;
50      state.leg[1].adhesion_b= state.leg[1].solenoid_b &&
state.leg[1].magnet;
        state.leg[1].adhesion_c= state.leg[1].solenoid_c &&
state.leg[1].magnet;

55    /*
        * Send it to the tragen task, which should be pended

```

SUBSTITUTE SHEET

- 53 -

```

on it receipt.
    * Since tragen is of lower priority than us, nothing
will happen
    * until we stop later
5      */

    /* send state message */
    if (msgQSend(
        sav_servo_tragen,
10      (char *)&state,
        sizeof(state),
        WAIT_FOREVER,
        MSG_PRI_NORMAL
        ) == _ERROR)
15      sav_panic("servo_task: error writing
servo->tragen queue\n");

    /* set the timeout a little longer than the servo
interval */
20      timeout = (period * sysClkRateGet())/1000000 + 1;

    /* wait for the tragen task to reply */
    if (msgQReceive(
        sav_tragen_servo,
25      (char *)&command,
        sizeof(command),
        timeout
        ) == _ERROR)
        sav_panic("servo_task: error reading
30      tragen->servo queue\n");

    /*
    * The message we got from the tragen has the next
35      command in it
    * but we don't do anything with it until the top of
the loop next
    * time around. Instead we check the servos are happy,
have not hit
40      * their limit switches, the power is still on, etc,
etc.
    */

45      /* check the walker still has power */
    if (io_get(IO_PM1) == 0)
        sav_panic("servo_task: +64VDC umbilical power
failed\n");
    #if 0
50      if (io_get(IO_PM2) == 0)
        sav_panic("servo_task: +32VDC umbilical power
failed\n");
    #endif

55      /* for each of the servo channels */
    for (channel = 0; channel < NSERVO; ++channel) {

```

SUBSTITUTE SHEET

- 54 -

```

        word signal;

        /* velocity servo inhibit indicators and limit
switches */
5      if (io_get(servo[channel].cw) == 0)
        sav_panic("servo_task: servo %d: CW limit
alarm\n", channel);
        if (io_get(servo[channel].ccw) == 0)
10     sav_panic("servo_task: servo %d: CCW limit
alarm\n", channel);
        if (io_get(servo[channel].inhibit) == 0)
        sav_panic("servo_task: servo %d: inhibit
alarm\n", channel);

15     /* position servo chip status bits */
        lm628_rdsigs(servo[channel].addr, &signal);

        if (signal & LM628_G_PERR)
        sav_panic(
20     "servo_task: servo %d: excessive position
alarm (%04x)\n",
        channel, signal
        );
        if (signal & LM628_G_WRAP)
25     sav_panic(
        "servo_task: servo %d: wraparound alarm
(%04x)\n",
        channel, signal
        );
30     if (signal & LM628_G_CERR)
        sav_panic(
        "servo_task: servo %d: command error alarm
(%04x)\n",
        channel, signal
35     );
    }

    /* that's all really */
40 }

    sav_panic("servo_task: servo loop exited\n");
}

45 /*
   * traj.c
   */
#ifdef VXWORKS
#include <vxWorks.h>
#include <msgQLib.h>
50 #include <semLib.h>
#include <stdioLib.h>
#else
#include <stdio.h>
55 #endif
#include <math.h>

```

SUBSTITUTE SHEET

- 55 -

```

#include "sav.h"
#include "tragen.h"
#include "traj.h"
5
#define SOLDOWN 1
#define SOLUP 0
#define MAGON 1
#define MAGOFF 0
10 #define RATE 0.02
#define rv 0.02 /* reset base speed */

#define TOFFSET 50.0 /* tool point offset
in x direction */
15

TRANS tool;
LEG leg1, leg0;
LEG *leg; /* the leg to which the fixed
20 coord. is * attached */
LEG *leg_up; /* the leg which is up */

int seg_no = 0; /* segment number executed */
25 int sts = NEWTAR, sts1 = NEWSTEP, sts2 = 1;
int get_tar = 1;
int home = 1; /* reset leg home status */
int rst_ct = 0; /* reset counter */
double vd_a = 0.0, rt_a[2] = {0.0, TOFFSET};
30

#ifdef VXWORKS
SEM_ID traj_sem = NULL;
#endif

35
/*
* rt[]: desired target vector in the vehicle coord. sys.
* vd: desired velocity m/s
*/
40 double vd = 0.0, rt[2] = {0.0, TOFFSET};

double rtt[2]; /* vector of the target
point in the fixed leg
* coord. system */
45 double ttf[3][3], tt0[3][3], tt1[3][3], tf[3][3];
double rst_l, rst_s;
FILE *fp;
FILE *fpv;

50
traj_init(mode)
int mode;
{
int i, j;
55
#ifdef VXWORKS

```

SUBSTITUTE SHEET

- 56 -

```

    traj_sem = semMCreate(0);
#endif

    /* initialise tool transform */
5   for (i = 0; i < 3; i++)
        for (j = 0; j < 3; j++) {
            tool.t[i][j] = 0.0;
            ttf[i][j] = 0.0;
            if (i == j)
10         tool.t[i][j] = ttf[i][j] = 1.0;
        }
    tool.t[1][2] = TOFFSET;

    col_init();
15 }

traj(svact, svcom)
20   savmsg_servo_tragen *svact;
    savmsg_tragen_servo *svcom;
{
    int    i, j;
    int    jx, jy;
25   static int jxp, jyp;

    if (svact->serial == 0) {
        /* initialise leg econdor counters and transformas */
        leg0.encs = svact->position[0];
30     leg0.encl = svact->position[1];
        leg1.encl = svact->position[2];
        leg1.encs = svact->position[3];
        enc_axis(&leg1);
        enc_axis(&leg0);
35     trans_mat(-25.0, -leg1.al, atan2(leg1.al - leg1.as,
        ARMLen), leg1.t);
        trans_mat(25.0, -leg0.al, atan2(leg0.as - leg0.al,
        ARMLen), leg0.t);
40     /* initialise leg status */
        if (svact->leg[0].adhesion_a &&
        svact->leg[0].adhesion_b &&
        svact->leg[0].adhesion_c) {
45         leg0.status = LEGDOWN;
        leg1.status = LEGUP;
        svcom->leg[0].magnet = MAGON;
        svcom->leg[1].magnet = MAGOFF;
        } else if (svact->leg[1].adhesion_a &&
50   svact->leg[1].adhesion_b &&
        svact->leg[1].adhesion_c) {
        leg1.status = LEGDOWN;
        leg0.status = LEGUP;
        svcom->leg[1].magnet = MAGON;
55     svcom->leg[0].magnet = MAGOFF;
        } else {

```

- 57 -

```

        svcom->leg[0].solenoid_a =
svcom->leg[0].solenoid_b
        = svcom->leg[0].solenoid_c = SOLDOWN;
        traj_delay(10000);
5         leg0.status = LEGDOWN;
        leg1.status = LEGUP;
        svcom->leg[0].magnet = MAGON;
        svcom->leg[1].magnet = MAGOFF;
    }
10     svcom->leg[1].solenoid_a = svcom->leg[1].solenoid_b
        = svcom->leg[1].solenoid_c = SOLUP;
        svcom->leg[0].solenoid_a = svcom->leg[0].solenoid_b
        = svcom->leg[0].solenoid_c = SOLUP;
        leg1.name = LEG1;
15     leg0.name = LEG0;
        if (leg1.status == LEGDOWN) {
            leg = &leg1;
            leg_up = &leg0;
        } else if (leg0.status == LEGDOWN) {
20         leg = &leg0;
            leg_up = &leg1;
        }
    }

25     get_tar = sts == NEWTAR && (seg_no < leg->nc || leg->nc
== 0) && home;
        if (get_tar || sts1 == NEWSTEP)
            /* There is a new target and is not in the middle of
changeover phase */
30         get_newtar();

        if (traj_tar() == -1)
            sav_panic("Wrong target !!\n");

35     /* output commanded encoder counters */
        svcom->position[0] = leg0.encs;
        svcom->position[1] = leg0.encl;
        svcom->position[2] = leg1.encl;
        svcom->position[3] = leg1.encs;
40
        /* output commands for pnuemetics and magnets */
        if (seg_no == leg->no && vd != 0.0)
            if (leg->name == LEG0)
                svcom->leg[1].solenoid_a =
45     svcom->leg[1].solenoid_b
                    = svcom->leg[1].solenoid_c = SOLDOWN;
            else
                svcom->leg[0].solenoid_a =
        svcom->leg[0].solenoid_b
50         = svcom->leg[0].solenoid_c = SOLDOWN;

        if (seg_no == leg->nm && vd != 0.0) {
55     /* energise the magnet and start adhesion detection */
        svcom->leg[0].magnet = MAGON;

```

SUBSTITUTE SHEET

- 58 -

```

    svcom->leg[1].magnet = MAGON;
}

    if ((svacl->leg[0].magnet == MAGON) &&
5  (svacl->leg[1].magnet == MAGON)
        && seg_no == leg->ns && vd != 0.0) {
        svcom->leg[1].solenoid_a = svcom->leg[1].solenoid_b
        = svcom->leg[1].solenoid_c = SOLUP;
        svcom->leg[0].solenoid_a = svcom->leg[0].solenoid_b
10      = svcom->leg[0].solenoid_c = SOLUP;
        if (leg->name == LEG0) {
            svcom->leg[0].solenoid_a =
            svcom->leg[0].solenoid_b
            = svcom->leg[0].solenoid_c = SOLDOWN;
15          svcom->leg[0].magnet = MAGOFF;
        } else {
            svcom->leg[1].solenoid_a =
            svcom->leg[1].solenoid_b
            = svcom->leg[1].solenoid_c = SOLDOWN;
20          svcom->leg[1].magnet = MAGOFF;
        }
    }

25  if ((seg_no == leg->nu) && vd != 0.0) {
        /* leg up phase */
        svcom->leg[1].solenoid_a = svcom->leg[1].solenoid_b
        = svcom->leg[1].solenoid_c = SOLUP;
        svcom->leg[0].solenoid_a = svcom->leg[0].solenoid_b
30      = svcom->leg[0].solenoid_c = SOLUP;
    }

    if ((seg_no == leg->nn) && vd != 0.0) {
        /* one step finished and leg reset starts */
35      leg1.status = !leg1.status;
        leg0.status = !leg0.status;
        if (leg1.status == LEGDOWN) {
            leg = &leg1;
            leg_up = &leg0;
40        } else if (leg0.status == LEGDOWN) {
            leg = &leg0;
            leg_up = &leg1;
        }
    }
45 }

traj_tar()
{
50     static int tl_seg;
        static double t0[3][3];    /* transform of fixed leg at
the begining of
                                * each step */
        static double sg_ang;      /* angle of the straight
55 line step */
        static int f_nn;

```

SUBSTITUTE SHEET

- 59 -

```

double rn[2];          /* vector of the next set point
*/
int i, j, seg_tmp;
double tt, ti[3][3];
5   if (vd == 0.0)
    return;

    if (stsl == NEWSTEP || get_tar) {
10   /* move toward a set point when change step or get
target is set */

        for (i = 0; i <= 2; i++)
            for (j = 0; j <= 2; j++)
15   /* keep the transform of fixed leg at beginning of
each step */
                t0[i][j] = leg->t[i][j];

        sts = WITHINTAR;
20   stsl = WITHINSTEP;
        mat_inv(ti, leg->t);
        transf(ti, rtt, rt); /* keep the target point in
fixed leg coord.

                                * sys. at the beginning of each step
25 */
        f_nn = traj_stck(rn);

        /* check if it is within the mobility range if not
return a new point
30   * rn(x, y) alone the line in vel. coord. sys. and
within the range
        * and flag the a new set point has been issued */

        seg_tmp = traj_seg(rn, &sg_ang); /* Trajectory
35 segmentation,

                                * returns total segment
                                * number */

        if (seg_tmp > 0) {
            tl_seg = seg_tmp; /* start moving towards a new
40 set point */
            seg_no = 0;
        } else {
            seg_no = leg->nc - 1; /* start catching up
NOW! the next

45   * set point will be taken care of
in
                                * the new step */

            f_nn = 0;
        }
50   }
    traj_step(tl_seg, f_nn, t0, sg_ang);

    if (f_nn == 0 && stsl == NEWSTEP)
55   sts = NEWTAR;
}

```

SUBSTITUTE SHEET

- 60 -

```

    traj_step(tl_seg, f_ns, t, sang)
        int    tl_seg, f_ns;
        double t[3][3], sang;
5   {
        double sp[2];          /* set point in fixed leg coord.
        sys. */
        double rtp[2];          /* tmp vector */
        double ti[3][3];        /* inverted transform of
10  currently fixed leg */

        seg_no = seg_no + 1;
        if (seg_no < leg->no)
            traj_gtst(t, sang, seg_no, sp);    /* get next set
15  point */
        traj_axis(&leg1, sp);
        traj_axis(&leg0, sp);

        if (seg_no == tl_seg) {
20          sts1 = NEWSTEP;          /* one step completed */
          sts2 = NEWSTEP;          /* one step completed */
          rst_ct = 0;
          rst_l = leg->al;          /* axial displacement of last
25  step (large
          * motor) */
          rst_s = leg->as;          /* axial displacement of last
          step (small
          * motor) */

          if (f_ns)
30          transf(leg->t, rt, rtt); /* convert the target
          point in the
          * new vehicle coord. sys. */
          if (leg->name == LEG1)
              leg = &leg0;
35          else
              leg = &leg1;
        }
    }

40  traj_stck(r)
    double r[2];
    {
        double rtp[2], ri[2], ti[3][3];
45        double k_stp, k_ang, nd;
        int    f_rg = 0, f_n = 0;

        mat_inv(ti, leg->t);
        transf(ti, ri, rt);
50        r[0] = rt[0];
        r[1] = rt[1];
        nd = 2.0;
        do {
            if (inv_kin(ri, &k_stp, &k_ang) == -1)
55            sav_panic("traj_stck: inv_kin error\n");

```

SUBSTITUTE SHEET

- 61 -

```

/* if out of range, return 1 */
f_rg = range_check(k_stp, k_ang);
if (f_rg) {
    r[0] = (rt[0] - tool.t[0][2]) / nd + tool.t[0][2];
5    r[1] = (rt[1] - tool.t[1][2]) / nd + tool.t[1][2];
    f_n = 1;
}
nd = nd + 1.0;
transf(ti, ri, r);
10 } while (f_rg);
return f_n;
}

15 traj_seg(r, ang)
    double r[2], *ang;
{
    int i, sn;
    double l; /* distance between tool point and
20 target point */
    double n;

    l = sqrt((r[0] - tool.t[0][2]) * (r[0] - tool.t[0][2])
        + (r[1] - tool.t[1][2]) * (r[1] - tool.t[1][2]));
25 sn = (int) ((l / (vd * 1000)) / RATE + 0.5); /* number
of segments */
    if ((sn - CTHTIME - OVLPTIME - ADHTIME - MGTIME -
        LGUPTIME) <= 0)
        return (-1);
30 /* when the new set point's demands are beyond the
reach of currently
* clamping leg, start changeover with the old data */

    leg1.nc = leg0.nc = sn - CTHTIME - OVLPTIME - ADHTIME -
35 MGTIME - LGUPTIME;
    leg1.no = leg0.no = sn - OVLPTIME - ADHTIME - MGTIME -
        LGUPTIME;
    leg1.nm = leg0.nm = sn - ADHTIME - MGTIME - LGUPTIME;
    leg1.ns = leg0.ns = sn - MGTIME - LGUPTIME;
40 leg1.nu = leg0.nu = sn - LGUPTIME;
    leg1.nn = leg0.nn = sn;
    *ang = PI / 2.0 - atan2(r[1] - tool.t[1][2], r[0] -
        tool.t[0][2]);

45 return sn;
}

traj_gtst(t, ang, n, p)
50 double t[3][3], ang, p[2];
    int n;
{
    double l, rtp[2], ti[3][3];

55 l = RATE * vd * 1000 * n; /* length of the straight
line covered by the

```

- 62 -

```

                                * end of next segment */
    rtp[0] = 1 * sin(ang) + tool.t[0][2];
    rtp[1] = 1 * cos(ang) + tool.t[1][2];
    mat_inv(ti, t);
5   transf(ti, p, rtp);          /* present next segment
point in fixed leg
                                * coord. sys. */
}

10  traj_axis(lg, p)
    LEG *lg;
    double p[2];
{
15     if (lg->status == LEGDOWN && seg_no < lg->no)
        traj_down(p);          /* leg act as the one on the
surface */

    if (lg->status == LEGUP && seg_no <= lg->nc)
20     traj_reset(lg);          /* leg resetting */

    if (lg->status == LEGUP && seg_no >= lg->nc && seg_no <
lg->no)
    traj_catch(lg);            /* leg catching up */
25     if (seg_no >= lg->no && seg_no <= lg->nn)
        traj_overlap(lg);      /* leg in over-lapping phase */

    trans_mat(-25.0, -leg1.al, atan2(leg1.al - leg1.as,
30  ARMLEN), leg1.t);
    trans_mat(25.0, -leg0.al, atan2(leg0.as - leg0.al,
ARMLEN), leg0.t);
    col_cllsn(leg1.al, leg1.as, leg0.al, leg0.as);
}
35

    traj_down(r)
    double r[2];
{
40     double s, a;

    if (inv_kin(r, &s, &a) == -1)
        sav_panic("traj_down: inv_kin error\n");
    leg_axis(s, a, leg);
45     axis_enc(leg);
}

    traj_overlap(lg)
50     LEG *lg;
{
    double st, an;

    lg->al = lg->al + vd * RATE * 1000; /* v--m/s and
55  st--mm */
    lg->as = lg->as + vd * RATE * 1000; /* v--m/s and

```

SUBSTITUTE SHEET

- 63 -

```

st--mm */
    axis_enc(lg);
}

5   traj_reset(lg)
    LEG    *lg;
    {
10      double  ss, sl;

        if (lg->al <= 0.05 * rst_l || lg->al >= 0.95 * rst_l)
            sl = lg->al - 1.00 * rv * RATE * 1000;
        else if (lg->al <= 0.1 * rst_l || lg->al >= 0.9 *
15      rst_l)
            sl = lg->al - 1.25 * rv * RATE * 1000;
        else if (lg->al <= 0.15 * rst_l || lg->al >= 0.85 *
            rst_l)
            sl = lg->al - 1.5 * rv * RATE * 1000;
        else if (lg->al <= 0.2 * rst_l || lg->al >= 0.8 *
20      rst_l)
            sl = lg->al - 2.0 * rv * RATE * 1000;
        else
            sl = lg->al - RTSPEED * rv * RATE * 1000;

25      if (lg->as <= 0.05 * rst_s || lg->as >= 0.95 * rst_s)
            ss = lg->as - 1.00 * rv * RATE * 1000;
        else if (lg->as <= 0.1 * rst_s || lg->as >= 0.9 *
            rst_s)
            ss = lg->as - 1.25 * rv * RATE * 1000;
30      else if (lg->as <= 0.15 * rst_s || lg->as >= 0.85 *
            rst_s)
            ss = lg->as - 1.5 * rv * RATE * 1000;
        else if (lg->as <= 0.2 * rst_s || lg->as >= 0.8 *
            rst_s)
35      ss = lg->as - 2.0 * rv * RATE * 1000;
        else
            ss = lg->as - RTSPEED * rv * RATE * 1000;

        if (sl <= 0.0)
40      sl = 0.0;
        if (ss <= 0.0)
            ss = 0.0;
        if (ss == 0.0 || sl == 0.0)
            home = 1;
45      else
            home = 0;

        lg->al = sl;
        lg->as = ss;
50      axis_enc(lg);
    }

    traj_catch(lg)
55      LEG    *lg;
    {

```

SUBSTITUTE SHEET

- 64 -

```

        double st, an;
        int nco;

        if ((nco = lg->no - seg_no) != 1)
5         lg->al = lg->as = lg->al + vd * RATE * 1000 / nco;
        else
            lg->al = lg->as = lg->al + vd * RATE * 1000;

        axis_enc(lg);
10    }

    traj_stop()
    {
15        leg->al = leg->al;
        leg->as = leg->as;
        axis_enc(leg);
    }

20    traj_check(lg)
        LEG *lg;
    {
        double tti[3][3];
25        trans_mat(-25.0, -leg1.al - TOFFSET, atan2(leg1.al -
leg1.as, ARMLEN), tt1);
        trans_mat(25.0, -leg0.al - TOFFSET, atan2(leg0.as -
leg0.al, ARMLEN), tt0);
30        if (sts2) {
            if (lg->name == LEG0)
                mat_mult(tf, ttf, tt0);
            else
                mat_mult(tf, ttf, tt1);
35        sts2 = 0;
        }
        if (lg->name == LEG0)
            mat_inv(tti, tt0);
        else
40        mat_inv(tti, tt1);
        mat_mult(ttf, tf, tti);
        fprintf(fp, "%f %f\n", ttf[0][2], ttf[1][2]);
    }

45    vel_check()
    {
        static double a0 = 0.0, a1 = 0.0, a2 = 0.0, a3 = 0.0;
        double v0, v1, v2, v3;
50        static int sno = 0;

        v0 = (leg0.as - a0) / RATE;
        v1 = (leg0.al - a1) / RATE;
        v2 = (leg1.al - a2) / RATE;
55        v3 = (leg1.as - a3) / RATE;
        fprintf(fpv, "%d %f %f %f %f\n", sno, v0, v1, v2, v3);
    }

```

SUBSTITUTE SHEET

- 65 -

```

        a0 = leg0.as;
        a1 = leg0.al;
        a2 = leg1.al;
        a3 = leg1.as;
5      sno++;
    }

leg_axis(s, a, lg)
10    double s, a;
    LEG *lg;
    {
        lg->al = s;
        if (lg->name == LEG1)
15        lg->as = lg->al + tan(a) * ARMLEN;
        else
            lg->as = lg->al - tan(a) * ARMLEN;
    }

20    axis_enc(lg)
        LEG *lg;
    {
        lg->encl = (int) (lg->al * ENC_L / SCW_P);
25        lg->encs = (int) (lg->as * ENC_L / SCW_P);
    }

    enc_axis(lg)
30    LEG *lg;
    {
        lg->al = lg->encl / ENC_L * SCW_P;
        lg->as = lg->encs / ENC_L * SCW_P;
    }
35

    get_newtar()
    {
        #ifdef VXWORKS
40        semTake(traj_sem, WAIT_FOREVER);
        #endif
        vd = vd_a;
        rt[0] = rt_a[0];
        rt[1] = rt_a[1];
45        /* #ifdef VXWORKS semGive(traj_sem); #endif */
        return 1;
    }

50    traj_settar(v, x, y)
        double v, x, y;
    {
        #ifdef VXWORKS
            semTake(traj_sem, WAIT_FOREVER);
55        #endif
        vd_a = v;

```

SUBSTITUTE SHEET

- 66 -

```

    rt_a[0] = x + tool.t[0][2];
    rt_a[1] = y + tool.t[1][2];
    sts = NEWTAR;
    /* #ifdef VXWORKS semGive(traj_sem); #endif */
5  }

    traj_delay(n)
    int    n;
10 {
    int    i;

    for (i = 0; i <= n; i++);
15 }

    /*
    * math.c
    */
    #ifdef VXWORKS
20 #include <vxWorks.h>
    #include <stdioLib.h>
    #else
    #include <stdio.h>
    #endif
25 #include <math.h>

    trans_mat(x, y, al, t)
    double x, y, al, t[3][3];
30 {
    t[0][0] = cos(al);
    t[0][1] = -sin(al);
    t[0][2] = x;
    t[1][0] = sin(al);
35 t[1][1] = cos(al);
    t[1][2] = y;
    t[2][0] = 0.0;
    t[2][1] = 0.0;
    t[2][2] = 1.0;
40 }

    transf(t, p, ip)
    double t[3][3], p[2], ip[2];
45 {
    double v[3], v1[3];

    v[0] = ip[0];
50 v[1] = ip[1];
    v[2] = 1.0;
    mat_vec_multi(t, v, v1, 3, 3);
    p[0] = v1[0];
    p[1] = v1[1];
55 }

```

SUBSTITUTE SHEET

- 67 -

```

mat_vec_multi(m, v1, v2, n1, n2)
double m[3][3], v1[3], v2[3];
int n1, n2;
{
5   int i, j;

   for (i = 0; i < n1; i++) {
       v2[i] = 0.0;
       for (j = 0; j < n2; j++) {
10          v2[i] = m[i][j] * v1[j] + v2[i];
       }
   }
}

15 mat_mult(m, m1, m2)
double m[3][3], m1[3][3], m2[3][3];
{
   int i, j, k;
20  double tmp = 0.0;

   for (i = 0; i < 3; i++)
       for (j = 0; j < 3; j++) {
           for (k = 0; k < 3; k++) {
25              tmp = tmp + m1[i][k] * m2[k][j];
              m[i][j] = tmp;
           }
           tmp = 0.0;
       }
30 }

mat_inv(mi, m)
double mi[3][3], m[3][3];
35 {
   double d, mm[3][3];
   int i, j;

   d = m[0][0] * m[1][1] * m[2][2] + m[0][1] * m[1][2] *
40  m[2][0]
   + m[1][0] * m[2][1] * m[0][2] - m[0][2] * m[1][1] *
   m[2][0]
   - m[0][1] * m[1][0] * m[2][2] - m[1][2] * m[2][1] *
45  m[0][0];

   mm[0][0] = m[1][1] * m[2][2] - m[1][2] * m[2][1];
   mm[0][1] = -(m[1][0] * m[2][2] - m[1][2] * m[2][0]);
   mm[0][2] = m[1][0] * m[2][1] - m[1][1] * m[2][0];
   mm[1][0] = -(m[0][1] * m[2][2] - m[0][2] * m[2][1]);
50  mm[1][1] = m[0][0] * m[2][2] - m[0][2] * m[2][0];
   mm[1][2] = -(m[0][0] * m[2][1] - m[0][1] * m[2][0]);
   mm[2][0] = m[0][1] * m[1][2] - m[0][2] * m[1][1];
   mm[2][1] = -(m[0][0] * m[1][2] - m[0][2] * m[1][0]);
55  mm[2][2] = m[0][0] * m[1][1] - m[0][1] * m[1][0];

   for (i = 0; i < 3; i++)

```

SUBSTITUTE SHEET

- 68 -

```

        for (j = 0; j < 3; j++)
            mi[i][j] = mm[j][i] / d;
    }

5  /*
   * kine.c
   */
   #ifdef VXWORKS
   #include <vxWorks.h>
10  #include <stdioLib.h>
   #else
   #include <stdio.h>
   #endif
   #include <math.h>
15  #include "traj.h"

inv_kin(r, step, angle)
    double r[2];
20    double *step, *angle;
{
    extern TRANS tool;
    extern LEG *leg;
    double s, ang;
25    double str;

    str = r[0] * r[0] + r[1] * r[1] - SEP * SEP;
    if (str < 0.0) {
        return -1;
30    }
    s = sqrt(str) - tool.t[1][2];
    if (leg->name == LEG1)
        ang = asin((SEP * r[1] - r[0] * (s + tool.t[1][2])) /
                    ((s + tool.t[1][2]) * (s + tool.t[1][2]) + SEP
35 * SEP));
    else
        ang = asin(-(SEP * r[1] + r[0] * (s + tool.t[1][2])) /
                    ((s + tool.t[1][2]) * (s + tool.t[1][2]) + SEP
40 * SEP));

    *step = s;
    *angle = ang;
}

45
for_kin(r, leg, stp, ang)
    double r[2];
    LEG *leg;
    double stp, ang;
50 {
    extern TRANS tool;

    if (leg->name == LEG1) {
        r[0] = SEP * cos(ang) - (stp + tool.t[1][2]) *
55 sin(ang);
        r[1] = SEP * sin(ang) + (stp + tool.t[1][2]) *

```

SUBSTITUTE SHEET

- 69 -

```

cos(ang);
    } else {
        r[0] = -SEP * cos(ang) - (stp + tool.t[1][2]) *
sin(ang);
5      r[1] = -SEP * sin(ang) + (stp + tool.t[1][2]) *
cos(ang);
    }
}

10 range_check(stp_lth, ang)
    double stp_lth, ang;
{
    double a;
15    ang = ang * 180.0 / PI;

    if (stp_lth >= 75.0)
        return 1;
20    if (stp_lth < 0.0) {
        sav_panic("Negetive step length\n");
        return -1;
    }
25    if (stp_lth >= 30.0 && stp_lth < 75.0)
        if (ang >= 0.0)
            a = 10.7 - 0.14267 * stp_lth;
        else
            a = -10.7 + 0.14267 * stp_lth;
30    else if (ang >= 0.0)
        a = 12.5 - 1.0 / 6.0 * stp_lth;
    else
        a = -12.5 + 1.0 / 6.0 * stp_lth;
    if (fabs(ang) >= fabs(a))
35    return 1;
    else
        return 0;
}

40 /*
   * sav.c -- system functions
   */
#include <vxWorks.h>
#include <taskLib.h>
45 #include <msgQLib.h>
#include <stdioLib.h>
#include <vme.h>
#include <stdarg.h>

50 #include "sav.h"
#include "servo.h"
#include "tragen.h"
#include "logger.h"
#include "io.h"
55

/* the sav's two main tasks */

```

SUBSTITUTE SHEET

- 70 -

```

int sav_task_servo = 0,
  sav_task_tragen = 0,
  sav_task_logger = 0;

5  /* message queue ID's */
MSG_Q_ID
  sav_servo_tragen = NULL,
  sav_tragen_servo = NULL,
  sav_tragen_logger = NULL;

10 static
  sav_cleanup(void),
  sav_start(void);

15 int sav_init(int mode)
  /*
   * The Entry Point: Starts up the walker software and
  returns
   *
  20   * Performs all the software initialisation.
   */
  {
    /* clean up the remnants of any previous run */
  25   sav_cleanup();

    /* initialise DIO system driver */
    io_init();

  30   /* initialise servos, switch on power and calibrate if
  required */
    servo_init(mode = SERVO_COLD);
    servo_power(1);

  35   if (mode == SERVO_COLD) {
    /*
     * Calibration must be done with one leg up. If two
     are down,
     * we need to lift one; if none are down, we must
  40   lower one
     * (presumably this occurs only on the flat!)
     */
    int leg0 = (
      (io_readback(IO_MENO) == MAG_ON) &&
  45      (io_readback(IO_SOLOA) == SOL_DOWN) &&
      (io_readback(IO_SOLOB) == SOL_DOWN) &&
      (io_readback(IO_SOLOC) == SOL_DOWN)
    );
    int leg1 = (
  50      (io_readback(IO_MEN1) == MAG_ON) &&
      (io_readback(IO_SOL1A) == SOL_DOWN) &&
      (io_readback(IO_SOL1B) == SOL_DOWN) &&
      (io_readback(IO_SOL1C) == SOL_DOWN)
    );

  55   printf(

```

SUBSTITUTE SHEET

- 71 -

```

        "sav_init: Leg 0 is %s, Leg 1 is %s\n",
        leg0 ? "down" : "up",
        leg1 ? "down" : "up"
    );
5
    /* if one leg down, one leg up, use the one that's
down */
    if ((leg0 || leg1) && !(leg0 && leg1)) {
10
        /* just tidy up */
        io_set(IO_MEN0, leg0 ? MAG_ON : MAG_OFF);
        io_set(IO_SOLOA, leg0 ? SOL_DOWN : SOL_UP);
        io_set(IO_SOLOB, leg0 ? SOL_DOWN : SOL_UP);
        io_set(IO_SOL0C, leg0 ? SOL_DOWN : SOL_UP);
15
        io_set(IO_MEN1, leg1 ? MAG_ON : MAG_OFF);
        io_set(IO_SOL1A, leg1 ? SOL_DOWN : SOL_UP);
        io_set(IO_SOL1B, leg1 ? SOL_DOWN : SOL_UP);
        io_set(IO_SOL1B, leg1 ? SOL_DOWN : SOL_UP);
20
    } else {
        /* if both legs are down, or no legs are down */

        /* put leg 0 down */
25
        io_set(IO_MEN0, MAG_ON);
        io_set(IO_SOLOA, SOL_DOWN);
        io_set(IO_SOLOB, SOL_DOWN);
        io_set(IO_SOL0C, SOL_DOWN);

        /* and pull leg 1 up */
30
        io_set(IO_MEN1, MAG_OFF);
        io_set(IO_SOL1A, SOL_UP);
        io_set(IO_SOL1B, SOL_UP);
        io_set(IO_SOL1C, SOL_UP);
35
    }
    taskDelay(30);

    /* now we can calibrate */
    servo_calib();
40
}

/* mark that the hardware is initialised */
io_set(IO_WARM, 1);
45

/* initialise trajectory generator software */
traj_init();

/* launch tragen and servo tasks */
50
return sav_start();
}

static sav_cleanup(void)
55
/*
    * delete remnants of old tasks and queues

```

SUBSTITUTE SHEET

- 72 -

```

    */
{
    int tid;

5    if (sav_task_servo)
        taskDelete(sav_task_servo);
    if (sav_task_tragen)
        taskDelete(sav_task_tragen);
    if (sav_task_logger)
10    taskDelete(sav_task_logger);

    if (sav_tragen_servo)
        msgQDelete(sav_tragen_servo);
    if (sav_servo_tragen)
15    msgQDelete(sav_servo_tragen);
    if (sav_tragen_logger)
        msgQDelete(sav_tragen_logger);

    /* delete any ghosts of older images */
20    if (tid = taskNameToId("servo"))
        taskDelete(tid);
    if (tid = taskNameToId("tragen"))
        taskDelete(tid);
    if (tid = taskNameToId("logger"))
25    taskDelete(tid);

    /* re-arrange vxWorks system tasks priority */
    if (tid = taskNameToId("tLogTask"))
        taskPrioritySet(tid, 25);
30    if (tid = taskNameToId("tShell"))
        taskPrioritySet(tid, 20);

    return 0;
}
35

static sav_start(void)
    /*
    * Create the real-time tasks and message queues
    */
{
    /* create servo->tragen message queue */
    if (! (sav_servo_tragen = msgQCreate(
40        2, sizeof(savmsg_servo_tragen), MSG_Q_FIFO
        ))) {
55    printErr("sav_start: can't build servo->tragen
        queue\n");
        return ERROR;
    }

    /* create tragen->servo message queue */
    if (! (sav_tragen_servo = msgQCreate(
        2, sizeof(savmsg_tragen_servo), MSG_Q_FIFO
        ))) {
55    sav_panic("sav_start: can't build servo->tragen
        queue\n");
    }
}

```

SUBSTITUTE SHEET

- 73 -

```

        return ERROR;
    }

    /* create tragen->logger message queue */
5    if (! (sav_tragen_logger = msgQCreate(
        100, sizeof(savmsg_tragen_logger), MSG_Q_FIFO
        ))) {
        printErr("sav_start: can't build servo->logger
10    queue\n");
        return ERROR;
    }

    /* create logger task */
    if (! (sav_task_logger = taskSpawn(
15    "logger", 15, VX_STDIO, 5000, logger_task, LOGFILE
        ))) {
        printErr("sav_start: can't spawn logger task\n");
        return ERROR;
    }

20    /* create tragen task */
    if (! (sav_task_tragen = taskSpawn(
        "tragen", 12, VX_FP_TASK, 5000, tragen_task, traj
        ))) {
25    printErr("sav_start: can't create tragen task\n");
        return ERROR;
    }

    /* create servo task */
30    if (! (sav_task_servo = taskSpawn(
        "servo", 10, VX_STDIO, 5000, servo_task, PERIOD
        ))) {
        printErr("sav_start: can't create servo task\n");
        return ERROR;
35    }

    return 0;
}

40    int sav_panic(char *format, ...)
    {
        int tid;
        va_list ap;
45    va_start(ap, format);

        /* print error message */
        printf("\n\007%s: ", taskName(0));
        vprintf(format, ap);
50    printf("%s: errno = ", taskName(0));
        printErrno(errnoGet(0));

        /* restore shell priority */
        if (tid = taskNameToId("tShell"))
55    taskPrioritySet(tid, 1);
    }

```

- 74 -

```

/* kill servo/magnet power */
servo_power(0);

/* freeze the system by stopping motion tasks */
5  if (taskIdSelf() == sav_task_servo) {
    if (sav_task_tragen)
        taskSuspend(sav_task_tragen);
    if (sav_task_servo)
        taskSuspend(sav_task_servo);
10  } else {
    if (sav_task_servo)
        taskSuspend(sav_task_servo);
    if (sav_task_tragen)
        taskSuspend(sav_task_tragen);
15  }

    return -1;
}

20 /*
   * tragen.c -- trajectory generator task
   */
#include <vxWorks.h>
#include <taskLib.h>
25 #include <msgQLib.h>
#include <stdioLib.h>

#include "sav.h"
#include "tragen.h"
30

int tragen_task(tragen_func trag)
/*
   * The tragen task runs synchronously with the servo
35 task. It awaits a
   * message from the servo task on the message queue;
   computes the next
   * servo/magnet demand with trag(), and sends it back
   to the servo
40   * task which feeds it to the servos/magnets/etc
   */
{
    savmsg_servo_tragen state;
    savmsg_tragen_servo command;
45    savmsg_tragen_logger log;
    unsigned timeout = WAIT_FOREVER;

    while (1) {
50        /* wait for a message from servo */
        if (msgQReceive(
            sav_servo_tragen,
            (char *)&state,
            sizeof(state),
55            timeout
        ) == ERROR)

```

SUBSTITUTE SHEET

- 75 -

```

        sav_panic("tragen_task: error reading
servo->tragen queue\n");

        /* set the timeout a little longer than the servo
5 interval */
        timeout = (state.period * sysClkRateGet())/1000000 +
1;

        /* compute next point */
10      traj(&state, &command);

        /* send command message to servo */
        if (msgQSend(
15          sav_tragen_servo,
          (char *)&command,
          sizeof(command),
          WAIT_FOREVER,
          MSG_PRI_NORMAL
20        ) == ERROR)
          sav_panic("tragen_task: error writing
tragen->servo queue\n");

        /* build message for logger */
25      memcpy(&log.state, &state, sizeof(state));
      memcpy(&log.command, &command, sizeof(command));

        /* send message to logger */
        if (msgQSend(
30          sav_tragen_logger,
          (char *)&log,
          sizeof(log),
          WAIT_FOREVER,
          MSG_PRI_NORMAL
35        ) == ERROR)
          sav_panic("tragen_task: error writing
tragen->logger queue\n");
    }
}
40
/*
 * utili.c
 */
#ifdef VXWORKS
45 #include <vxWorks.h>
#include <msgQLib.h>
#include <stdioLib.h>
#else
#include <stdio.h>
50 #endif
#include <math.h>

#include "sav.h"
#include "io.h"
55

```

- 76 -

```

int      dp = 500000, dpl = 20000;

ut_init()
{
5   /* initialise DIO system driver */
   io_init();

   /* Turn the walker power on */
   servo_power(1);
10  }

ut_step()
{
15   if (io_readback(IO_MEN0) == MAG_ON) {
       io_set(IO_SOL1A, SOL_DOWN);
       io_set(IO_SOL1B, SOL_DOWN);
       io_set(IO_SOL1C, SOL_DOWN);
       ut_dly(dp);
20   io_set(IO_MEN1, MAG_ON);
       ut_dly(dpl);
       io_set(IO_SOL1A, SOL_UP);
       io_set(IO_SOL1B, SOL_UP);
       io_set(IO_SOL1C, SOL_UP);
25   io_set(IO_MEN0, MAG_OFF);
       ut_dly(dpl);
       io_set(IO_SOLA, SOL_UP);
       io_set(IO_SLOB, SOL_UP);
       io_set(IO_SLOC, SOL_UP);
30   return;
   }
   io_set(IO_SOLA, SOL_DOWN);
   io_set(IO_SLOB, SOL_DOWN);
   io_set(IO_SLOC, SOL_DOWN);
35   ut_dly(dp);
   io_set(IO_MEN0, MAG_ON);
   ut_dly(dpl);
   io_set(IO_SOLA, SOL_UP);
   io_set(IO_SLOB, SOL_UP);
40   io_set(IO_SLOC, SOL_UP);
   io_set(IO_MEN1, MAG_OFF);
   ut_dly(dpl);
   io_set(IO_SOL1A, SOL_UP);
   io_set(IO_SOL1B, SOL_UP);
45   io_set(IO_SOL1C, SOL_UP);
}

ut_dly(number)
50   int      number;
   {
       int      i;

       for (i = 0; i <= number; i++);
55  }

```

- 77 -

```

ut_alldown()
{
    io_set(IO_SOLA, SOL_DOWN);
5   io_set(IO_SOLB, SOL_DOWN);
    io_set(IO_SOLC, SOL_DOWN);
    io_set(IO_SOL1A, SOL_DOWN);
    io_set(IO_SOL1B, SOL_DOWN);
10   io_set(IO_SOL1C, SOL_DOWN);
    ut_dly(dp);
    io_set(IO_MEN0, MAG_ON);
    io_set(IO_MEN1, MAG_ON);
    ut_dly(dpl);
15   io_set(IO_SOLA, SOL_UP);
    io_set(IO_SOLB, SOL_UP);
    io_set(IO_SOLC, SOL_UP);
    io_set(IO_SOL1A, SOL_UP);
    io_set(IO_SOL1B, SOL_UP);
20   io_set(IO_SOL1C, SOL_UP);
}

ut_allup()
{
25   io_set(IO_MEN0, MAG_OFF);
    io_set(IO_MEN1, MAG_OFF);
    ut_dly(dpl);
    io_set(IO_SOLA, SOL_UP);
    io_set(IO_SOLB, SOL_UP);
30   io_set(IO_SOLC, SOL_UP);
    io_set(IO_SOL1A, SOL_UP);
    io_set(IO_SOL1B, SOL_UP);
    io_set(IO_SOL1C, SOL_UP);
35   }

ut_free()
{
    if (io_readback(IO_MEN0) == MAG_ON) {
40   io_set(IO_MEN1, MAG_ON);
        io_set(IO_MEN0, MAG_OFF);
        io_set(IO_SOLA, SOL_UP);
        io_set(IO_SOLB, SOL_UP);
        io_set(IO_SOLC, SOL_UP);
45   return;
    }
    io_set(IO_MEN0, MAG_ON);
    io_set(IO_MEN1, MAG_OFF);
    io_set(IO_SOL1A, SOL_UP);
50   io_set(IO_SOL1B, SOL_UP);
    io_set(IO_SOL1C, SOL_UP);
}

55 ut_mcycle()
{

```

SUBSTITUTE SHEET

- 78 -

```

        io_set(IO_MEN0, MAG_ON);
        io_set(IO_MEN1, MAG_OFF);

        while (1) {
5           taskDelay(50);
            io_set(IO_MEN1, MAG_ON);

            taskDelay(50);
            io_set(IO_MEN0, MAG_ON);
10          taskDelay(50);
            io_set(IO_MEN1, MAG_OFF);

            taskDelay(50);
15          io_set(IO_MEN0, MAG_OFF);
        }
    }

    /*
20   * lm628.c -- National LM628 servo chip
    *
    * This is a basic chip interface library. Each chip
    command has a
    * function lm628_xxxx() which takes the base address of
25   the chip
    * (address of the command register). The data register is
    assumed to
    * be the byte at the next word address. All functions
    return the
30   * chip status byte.
    *
    * The library uses polling in the BUSY bit of the status
    register of
    * the LM628. In theory BUSY should never be high more than
35   100us, and
    * is generally only high for 15 to 25us.
    *
    * unsigned short is assumed to be two bytes
    */
40   #include <vxWorks.h>
    #include <msgQLib.h>
    #include <stdarg.h>

    #include "sav.h"
45   #include "lm628.h"

byte lm628_command(volatile byte *addr, byte command)
    /*
50   * Wait for BUSY bit to clear, push in command to
    LM628.
    * Again wait for BUSY bit to clear. Return status byte
    */
    {
55   volatile byte status = 0;

```

SUBSTITUTE SHEET

- 79 -

```

/* wait for busy bit to clear */
while ((status = *addr) & LM628_S_BUSY) ;

/* push command, unless it is RDSTAT */
5  if (command != LM628_C_RDSTAT)
    *addr = (command & 0xff);

/* wait for busy bit to clear again */
10 while ((status = *addr) & LM628_S_BUSY) ;

/* check for command error */
if (status & LM628_S_CERR)
    logMsg("lm628_command: 0x%08x: CERR 0x%02x\n", addr,
15  command);

    return status;
}

20 byte lm628_data_write(volatile byte *addr, unsigned nwords,
    word *data)
    /*
     * Write nwords of data to the LM628, polling the BUSY
    bit to
    25  * regulate transmission. Returns status byte
     */
    {
        volatile byte status = 0;
        volatile byte *daddr = addr + 2;
    30  unsigned count = 0;

        for (count = 0; count < nwords; ++count) {

            /* wait for busy bit to clear */
            35  while ((status = *addr) & LM628_S_BUSY) ;

            /* write the two bytes of the word */
            *daddr = (data[count] >> 8);
            *daddr = (data[count] & 0xff);
            40  }

            /* wait for busy bit to clear again */
            while ((status = *addr) & LM628_S_BUSY) ;

            /* check for command error */
            45  if (status & LM628_S_CERR)
                logMsg("lm628_data_write: 0x%08x: CERR\n", addr);

            return status;
    50  }

byte lm628_data_read(volatile byte *addr, unsigned nwords,
word *data)
55  /*
     * Read nwords of data to the LM628, polling the BUSY

```

SUBSTITUTE SHEET

- 80 -

```

    bit to
        * regulate reception. Returns status byte
        */
    {
5      volatile byte status = 0;
        volatile byte *daddr = addr + 2;
        unsigned count = 0;

10     for (count = 0; count < nwords; ++count) {

        /* wait for busy bit to clear */
        while ((status = *addr) & LM628_S_BUSY) ;

        /* read the two bytes of the word */
15     data[count] = ((word)(*daddr) << 8);
        data[count] |= (word)(*daddr);
    }

        /* wait for busy bit to clear again */
20     while ((status = *addr) & LM628_S_BUSY) ;

        /* check for command error */
        if (status & LM628_S_CERR)
            logMsg("lm628_data_read: 0x%08x: CERR\n", addr);
25     return status;
    }

30 /*
    * "utility" functions that should be macros
    */

    static int clip(int value, int lower, int upper)
35 {
        if (value > upper) return upper;
        if (value < lower) return lower;
        return value;
    }

40 static void split(word *data, int value)
    {
        data[0] = (word)(value >> 16);
        data[1] = (word)(value & 0xffff);
45 }

    static int glue(word *data)
    {
        return (((int)data[0]) << 16) | data[1];
50 }

    /*
    * "user" command functions, with (mostly) sanitized
55 arguments
    */

```

SUBSTITUTE SHEET

- 81 -

```

byte lm628_reset(volatile byte *addr)
{
    /* Reset LM628 */
    return lm628_command(addr, LM628_C_RESET);
5 }

byte lm628_port8(volatile byte *addr)
{
    /* Select 8-bit Output */
10    return lm628_command(addr, LM628_C_PORT8);
}

byte lm628_port12(volatile byte *addr)
{
15    /* Select 12-bit Output */
    return lm628_command(addr, LM628_C_PORT12);
}

byte lm628_dfh(volatile byte *addr)
20 {
    /* Define Home */
    return lm628_command(addr, LM628_C_DFH);
}

25 byte lm628_sip(volatile byte *addr)
{
    /* Set Index Position */
    return lm628_command(addr, LM628_C_SIP);
30 }

byte lm628_lpei(volatile byte *addr, word range)
{
    /* Load Position Error for Interrupt */
    lm628_command(addr, LM628_C_LPEI);
35    range &= 0x7fff;
    return lm628_data_write(addr, 1, &range);
}

byte lm628_lpes(volatile byte *addr, word range)
40 {
    /* Load Position Error for Stop */
    lm628_command(addr, LM628_C_LPES);
    range &= 0x7fff;
    return lm628_data_write(addr, 1, &range);
45 }

byte lm628_sbpa(volatile byte *addr, int position)
{
    /* Set Breakpoint, Absolute */
50    word data[2];

    lm628_command(addr, LM628_C_SBPA);
    split(data, clip(position, 0xc0000000, 0xffffffff));
    return lm628_data_write(addr, 2, data);
55 }

```

SUBSTITUTE SHEET

- 82 -

```

byte lm628_sbpr(volatile byte *addr, int position)
{
    /* Set Breakpoint, Relative */
    word data[2];
5    lm628_command(addr, LM628_C_SBPR);
    split(data, clip(position, 0xc0000000, 0x3fffffff));
    return lm628_data_write(addr, 2, data);
}
10 byte lm628_mski(volatile byte *addr, word imask)
{
    /* Mask Interrupts */
    lm628_command(addr, LM628_C_MSKI);
15    imask &= 0x007e;
    return lm628_data_write(addr, 1, &imask);
}

byte lm628_rsti(volatile byte *addr, word imask)
20 {
    /* Reset Interrupts. Note the mask is inverted! */
    lm628_command(addr, LM628_C_RSTI);
    imask = (~imask) & 0x007e;
25    return lm628_data_write(addr, 1, &imask);
}

byte lm628_lfil(volatile byte *ad, byte ds, word kp, word
ki, word kd, word il)
{
30    /* Load Filter Parameters */
    word data[5];

    /* we always load all parameters */
    lm628_command(ad, LM628_C_LFIL);
35    data[0] = ((word)ds << 8) | 0x0f;
    data[1] = kp;
    data[2] = ki;
    data[3] = kd;
    data[4] = il;
40    return lm628_data_write(ad, 5, data);
}

byte lm628_udf(volatile byte *addr)
{
45    /* Update Filter Parameters */
    return lm628_command(addr, LM628_C_UDF);
}

byte lm628_ltrj(volatile byte *addr, word tcw, ... /* int
50 a, int v, int p */)
{
    /* Load Trajectory */
    word data[7];
    unsigned nwords = 1;
55    va_list ap;
    va_start(ap, tcw);

```

- 83 -

```

        lm628_command(addr, LM628_C_LTRJ);
        data[0] = tcw;
        if (tcw & LM628_T_AL) {
            split(data + nwords, clip(va_arg(ap, int), 0,
5 0x3fffffff));
            nwords += 2;
        }
        if (tcw & LM628_T_VL) {
            split(data + nwords, clip(va_arg(ap, int), 0,
10 0x3fffffff));
            nwords += 2;
        }
        if (tcw & LM628_T_PL) {
            split(data + nwords, clip(va_arg(ap, int), 0xc0000000,
15 0x3fffffff));
            nwords += 2;
        }
        return lm628_data_write(addr, nwords, data);
    }

20 byte lm628_stt(volatile byte *addr)
    {
        /* Start Motion */
        return lm628_command(addr, LM628_C_STT);
25 }

byte lm628_rdstat(volatile byte *addr)
    {
        /* Read Status Byte */
30 return lm628_command(addr, LM628_C_RDSTAT);
    }

byte lm628_rdsigs(volatile byte *addr, word *sigs)
    {
35 /* Read Signals Register */
        lm628_command(addr, LM628_C_RDSIGS);
        return lm628_data_read(addr, 1, sigs);
    }

40 byte lm628_rdip(volatile byte *addr, int *position)
    {
        /* Read Index Position */
        word data[2];
        byte status;
45
        lm628_command(addr, LM628_C_RDIP);
        status = lm628_data_read(addr, 2, data);
        *position = glue(data);
        return status;
50 }

byte lm628_rddp(volatile byte *addr, int *position)
    {
        /* Read Desired Position */
55 word data[2];
        byte status;

```

SUBSTITUTE SHEET

- 84 -

```

        lm628_command(addr, LM628_C_RDDP);
        status = lm628_data_read(addr, 2, data);
        *position = glue(data);
        return status;
5   }

byte lm628_rdrp(volatile byte *addr, int *position)
{
    /* Read Real Position */
10   word data[2];
    byte status;

    lm628_command(addr, LM628_C_RDRP);
    status = lm628_data_read(addr, 2, data);
15   *position = glue(data);
    return status;
}

byte lm628_rddv(volatile byte *addr, int *velocity)
20 {
    /* Read Desired Velocity */
    word data[2];
    byte status;

25   lm628_command(addr, LM628_C_RDDV);
    status = lm628_data_read(addr, 2, data);
    *velocity = glue(data);
    return status;
}

30 byte lm628_rdrv(volatile byte *addr, short *velocity)
{
    /* Read Real Velocity */
    lm628_command(addr, LM628_C_RDRV);
35   return lm628_data_read(addr, 1, (word *)velocity);
}

byte lm628_rdsun(volatile byte *addr, short *isum)
{
40   /* Read Integration Sum */
    lm628_command(addr, LM628_C_RDSUM);
    return lm628_data_read(addr, 1, (word *)isum);
}

45 int lm628_show(volatile byte *addr)
{
    int position = 0, index = 0, desired = 0, error = 0,
    veldesired = 0;
50   short velocity = 0;
    word sigs;

    /* print header if no chip addr */
    if (! addr) {
55   printf("\n  %8s %6s %8s %10s %10s %-8s %8s %s\n",
        "addr",

```

- 85 -

```
5      "sigs",
      "index",
      "desired",
      "actual",
      "error",
      "vdes",
      "vel"
      );
      return 0;
10  }

      /* get chip info */
      lm628_rdsigs(addr, &sigs);
      lm628_rddp(addr, &desired);
15  lm628_rddp(addr, &desired);
      lm628_rddp(addr, &index);
      lm628_rdrp(addr, &position);
      lm628_rddv(addr, &veldesired);
      lm628_rdrv(addr, &velocity);
20  error = desired - position;

      /* print chip info to stdout */
      printf("0x%08x 0x%04x %8d %10d %10d %-8d %8d %d\n",
      addr,
      sigs,
25  index,
      desired,
      position,
      error,
      veldesired,
30  velocity
      );

      return 0;
35  }
```

SUBSTITUTE SHEET

- 86 -

```

#ifndef _lm628_h_
#define _lm628_h_
/*
 * lm628.h -- National LM628 servo chip
5 */

    /* user driver functions */
extern byte
    lm628_reset(volatile byte *addr),
10    lm628_port8(volatile byte *addr),
    lm628_port12(volatile byte *addr),
    lm628_dfh(volatile byte *addr),
    lm628_sip(volatile byte *addr),
    lm628_lpei(volatile byte *addr, word range),
15    lm628_lpes(volatile byte *addr, word range),
    lm628_sbpa(volatile byte *addr, int position),
    lm628_sbpr(volatile byte *addr, int position),
    lm628_mski(volatile byte *addr, word imask),
    lm628_rsti(volatile byte *addr, word imask),
20    lm628_lfil(volatile byte *ad, byte ds, word kp, word
    ki, word kd, word il),
    lm628_udf(volatile byte *addr),
    lm628_ltrj(volatile byte *addr, word tcw, ... /* int a,
    int v, int p */ ),
25    lm628_stt(volatile byte *addr),
    lm628_rdstat(volatile byte *addr),
    lm628_rdsigs(volatile byte *addr, word *sigs),
    lm628_rdip(volatile byte *addr, int *position),
    lm628_rddp(volatile byte *addr, int *position),
30    lm628_rdrp(volatile byte *addr, int *position),
    lm628_rddv(volatile byte *addr, int *velocity),
    lm628_rdrv(volatile byte *addr, short *velocity),
    lm628_rdsun(volatile byte *addr, short *isum);

35 extern int
    lm628_show(volatile byte *add);

    /* LM628 command bytes */
40 #define LM628_C_RESET      0x00      /* Reset LM628 */
    #define LM628_C_PORT8    0x05      /* Select 8-bit Output
    */
    #define LM628_C_PORT12   0x06      /* Select 12-bit Output
    */
45 #define LM628_C_DFH 0x02      /* Define Home */
    #define LM628_C_SIP 0x03      /* Set Index Position */
    #define LM628_C_LPEI     0x1b      /* Interrupt on Error */
    #define LM628_C_LPES     0x1a      /* Stop on Error */
    #define LM628_C_SBPA     0x20      /* Set Breakpoint,
50 Absolute */
    #define LM628_C_SBPR     0x21      /* Set Breakpoint,
    Relative */
    #define LM628_C_MSKI     0x1c      /* Mask Interrupts */
    #define LM628_C_RSTI     0x1d      /* Reset Interrupts */
55 #define LM628_C_LFIL     0x1e      /* Load Filter
    Parameters */

```

SUBSTITUTE SHEET

- 87 -

```

#define LM628_C_UDF 0x04      /* Update Filter Parameters
*/
#define LM628_C_LTRJ 0x1f      /* Load Trajectory */
#define LM628_C_STT 0x01      /* Start Motion */
5  #define LM628_C_RDSTAT 0xff  /* Read Status Byte */
#define LM628_C_RDSIGS 0x0c    /* Read Signals Register
*/
#define LM628_C_RDIP 0x09      /* Read Index Position
*/
10 #define LM628_C_RDDP 0x08     /* Read Desired Position
*/
#define LM628_C_RDRP 0x0a      /* Read Real Position */
#define LM628_C_RDDV 0x07      /* Read Desired Velocity
*/
15 #define LM628_C_RDRV 0x0b     /* Read Real Velocity */
#define LM628_C_RDSUM 0x0d     /* Read Integration Sum
*/

/* Filter Control Word: for LFIL command */
20 #define LM628_F_DMASK (0xff << 8) /* Derivative
Sample Mask */
#define LM628_F_DROLL 8        /* Derivative Sample
Roll */
#define LM628_F_KP (1 << 3)    /* Loading Kp data */
25 #define LM628_F_KI (1 << 2)  /* Loading Ki data */
#define LM628_F_KD (1 << 1)    /* Loading Kd data */
#define LM628_F_IL (1 << 0)    /* Loading Il data */

/* Trajectory Control Word: for LTRJ command */
30 #define LM628_T_FORW (1 << 12) /* Forward Direction
(Velocity Mode) */
#define LM628_T_VMODE (1 << 11) /* Velocity Mode */
#define LM628_T_SSTOP (1 << 10) /* Stop Smoothly */
#define LM628_T_STOP (1 << 9)  /* Stop Rapidly */
35 #define LM628_T_MOFF (1 << 8) /* Stop (Motor Off) */
#define LM628_T_AL (1 << 5)    /* Acceleration will be
Loaded */
#define LM628_T_ADR (1 << 4)   /* Acceleration Data is
Relative */
40 #define LM628_T_VL (1 << 3)  /* Velocity will be Loaded */
#define LM628_T_VDR (1 << 2)   /* Velocity Data is Relative
*/
#define LM628_T_PL (1 << 1)    /* Position will be Loaded */
#define LM628_T_PDR (1 << 0)   /* Position Data is Relative
45 */

/* Mask Byte: for MSKI/RSTI commands */
#define LM628_I_BDONE (1 << 6) /* Breakpoint Reached */
#define LM628_I_PERR (1 << 5)  /* Excessive Position
50 Error */
#define LM628_I_WRAP (1 << 4)  /* Wraparound Observed
*/
#define LM628_I_INDEX (1 << 3) /* Index Pulse Observed
*/
55 #define LM628_I_TDONE (1 << 2) /* Trajectory Complete
*/

```

SUBSTITUTE SHEET

- 88 -

```

#define LM628_I_CERR      (1 << 1)  /* Command Error */

    /* Status Byte: from RDSTAT command */
#define LM628_S_MOFF      (1 << 7)  /* Motor Off */
5  #define LM628_S_BDONE   (1 << 6)  /* Breakpoint Reached */
    #define LM628_S_PERR   (1 << 5)  /* Excessive Position
    Error */
    #define LM628_S_WRAP   (1 << 4)  /* Wraparound Observed
    */
10  #define LM628_S_INDEX   (1 << 3)  /* Index Pulse Observed
    */
    #define LM628_S_TDONE   (1 << 2)  /* Trajectory Complete
    */
    #define LM628_S_CERR    (1 << 1)  /* Command Error */
15  #define LM628_S_BUSY    (1 << 0)  /* Busy Bit */

    /* Signals Register */
#define LM628_G_HINT      (1 << 15) /* Host Interrupt */
#define LM628_G_ALNU      (1 << 14) /* Acceleration Loaded,
20 not updated */
    #define LM628_G_FLNU    (1 << 13) /* Filter Loaded, not
    updated */
    #define LM628_G_FORW    (1 << 12) /* Forward direction */
    #define LM628_G_VMODE   (1 << 11) /* Velocity Mode */
25  #define LM628_G_TARGET   (1 << 10) /* On Target */
    #define LM628_G_PERROFF (1 << 9)  /* Turn Off on Excessive
    Pos Error */
    #define LM628_G_PORT8   (1 << 8)  /* 8-bit Output Mode */
    #define LM628_G_MOFF    (1 << 7)  /* Motor Off */
30  #define LM628_G_BDONE   (1 << 6)  /* Breakpoint Reached */
    #define LM628_G_PERR    (1 << 5)  /* Excessive Position
    Error */
    #define LM628_G_WRAP    (1 << 4)  /* Wraparound Observed
    */
35  #define LM628_G_INDEX    (1 << 3)  /* Index Pulse Observed
    */
    #define LM628_G_TDONE   (1 << 2)  /* Trajectory Complete
    */
    #define LM628_G_CERR    (1 << 1)  /* Command Error */
40  #define LM628_G_ANI      (1 << 0)  /* Acquire Next Index */

#endif

#ifdef _servo_h_
45  #define _servo_h_

    /* LM628 does 3906 updates per second at 8MHz */
    #define SRATE (8000000/2048)

50  #define SERVO_WARM 0
    #define SERVO_COLD 1

extern int
    servo_init(int mode),
55  servo_calib(void),
    servo_power(int power),

```

SUBSTITUTE SHEET

- 89 -

```

        servo_task(unsigned long period),
        servo_show(void);

    #endif

5   #ifndef __joystick_h_
    #define __joystick_h_
    /*
10  * stuff for reading the control joystick
    */

    extern int joystick_read(int *x, int *y);
    extern int joystick_show(void);

15  #endif

    /*
    * traj.h
    */
20  #define R00x -191
    #define R00y 264
    #define R01x -191
    #define R01y -264
    #define R02x 191
25  #define R02y -264
    #define R03x 191
    #define R03y 264
    #define R10x 0
    #define R10y 0
30  #define R11x -100
    #define R11y 175
    #define R12x -100
    #define R12y -175
    #define R13x 125
35  #define R13y 0
    #define R20x 0
    #define R20y 0
    #define R21x 100
    #define R21y 175
40  #define R22x 100
    #define R22y -175
    #define R23x -125
    #define R23y 0
    #define R24x 25
45  #define R24y 112.5
    #define R25x 25
    #define R25y -112.5
    #define MARG 0

50  #define D10 155
    #define D11 155
    #define D12 80
    #define D20 135
    #define D21 135
55  #define D22 80
    #define D23 120

```

SUBSTITUTE SHEET

- 90 -

```

#define D24 120

#define SLO 10
#define SCO 12.5
5 #define SC1 50
#define SC2 37.5

#define PI 3.1415927
10 #define Pion2 1.5707963

#define VMAX_UP1 24000 /* small motor max. no load
speed (mm/s) */
#define VMAX_UP2 18000 /* large motor max. no load
15 speed (mm/s) */

#define AMAX_UP 129800 /* max. no load
acceleration (mm/s^2) */
#define ARMLEN 150.0 /* distance between small
20 and large motor
* axes */

#define SETNO 200 /* Max. number of set
points within a step */
25 #define LEGUP 0
#define LEGDOWN 1

#define LEG1 1
#define LEG0 2
30

#define NEWTAR 1
#define WITHINTAR 2
#define NEWSTEP 1
#define WITHINSTEP 2
35

#define SEP 25 /* seperation between leg
the vehicle centre */
#define RTSPEED 2.5 /* ratio between max. reset
speed and base
40 * speed */

#define ENC_L 2000 /* encoder lines per rev */
#define SCW_P 4 /* ball screw pitch (mm) */

45 #define CTHTIME 5 /* catch up time in terms
of number of
* intervals */
#define OVLPTIME 12 /* overlap time in terms of
number of
* intervals */
50 #define ADHTIME 5 /* toe travel time in terms
of number of
* intervals */
#define MGTIME 8 /* in terms of number of
55 intervals */
#define LGUPTIME 8 /* leg up time in terms of

```

SUBSTITUTE SHEET

- 91 -

```

number of intervals */

typedef struct _element {
    double *px;          /* new reference points */
5    double *py;          /* new reference points */
    double s;            /* object size (radius for
circles and half
                                * width for beams) */
                                /* object length (radius for
10    double d;
circles and half
                                * length for beams) */
    }      ELEMENT;

15    /* transforms */
    typedef struct _transform {
        double t[3][3];
    }      TRANS;

20    /* Leg structure */
    typedef struct _leg {
        double t[3][3];
25        int      status;
        int      nc;          /* leg catching-up start
time (in counts) */
        int      no;          /* leg over-laping start
time */
30        int      ns;          /* leg stepping start time
*/
        int      nm;          /* magnet detection start
time */
        int      nu;          /* leg up start time */
35        int      nn;          /* new fixed leg start time
*/
        int      ls;          /* leg status */
        int      name;        /* leg name ie leg1 or leg2 */
        int      encl, encs;   /* large and small motor
40    encoder counts */
        double al, as;        /* displacement alone the ball
screws */
    }      LEG;

45    typedef struct _setpnt {
        double x, y;
        struct _setpnt *next;
    }      SETPNT;

50    #ifndef _sav_h_
    #define _sav_h_
    /*
    * sav.h -- central header file
55    */

```

- 92 -

```
typedef unsigned char byte;
typedef unsigned short word;
typedef unsigned long lword;

5   /* number of servo axes we have */
   #define NSERVO 4

   /* servo/tragen period in us */
   #define PERIOD 20000
10
   /* logging file name */
   #define LOGFILE "LOG"

   /* startup modes */
15 #define INIT_WARM 0
   #define INIT_COLD 1

   /* inter-task message formats: */
20
   /* from servo to tragen */
   typedef struct {
       unsigned long serial, period;
       int position[NSERVO];
25   struct {
       byte adhesion_a;
       byte adhesion_b;
       byte adhesion_c;

30       byte magnet;
       byte solenoid_a;
       byte solenoid_b;
       byte solenoid_c;
       } leg[2];
35 } savmsg_servo_tragen;

   /* from tragen to servo */
   typedef struct {
       int position[NSERVO];
40   struct {
       byte magnet;
       byte solenoid_a;
       byte solenoid_b;
       byte solenoid_c;
45   } leg[2];
   } savmsg_tragen_servo;

   /* from tragen to logger */
   typedef struct {
50   savmsg_servo_tragen state;
       savmsg_tragen_servo command;
   } savmsg_tragen_logger;

55 #ifdef VXWORKS
       /* task ID's of system tasks */
```

SUBSTITUTE SHEET

- 93 -

```

extern int
    sav_task_servo,
    sav_task_tragen,
    sav_task_logger;
5
    /* message queue ID's */
extern MSG_Q_ID
    sav_servo_tragen,
    sav_tragen_servo,
10    sav_tragen_logger;
#endif

extern int
    sav_init(int mode),
15    sav_panic(char *format, ...);

#endif

#ifndef _tragen_h_
20 #define _tragen_h_
/*
    * tragen.h -- trajectory generator task
    */

25 typedef int (*tragen_func)(savmsg_servo_tragen *in,
    savmsg_tragen_servo *out);

extern int tragen_task(tragen_func trag);
extern int cos_traj(savmsg_servo_tragen *in,
30 savmsg_tragen_servo *out);
extern int traj(savmsg_servo_tragen *in,
    savmsg_tragen_servo *out);

#endif

35
#ifndef _vipc610_h_
#define _vipc610_h_
/*
    * vipc610.h -- GreenSpring VIPC610 IndustryPack Carrier
40    */

    /* VME short IO space */
#define VIPC610_IOBASE    0x6000

45    /* IPAC base addresses */
#define VIPC610_IPA      (void *) (VIPC610_IOBASE + 0x0000)
#define VIPC610_IPB      (void *) (VIPC610_IOBASE + 0x0100)
#define VIPC610_IPC      (void *) (VIPC610_IOBASE + 0x0200)
#define VIPC610_IPD      (void *) (VIPC610_IOBASE + 0x0300)
50

    /* IPAC ID addresses */
#define VIPC610_IDA      (void *) (VIPC610_IOBASE + 0x0080)
#define VIPC610_IDB      (void *) (VIPC610_IOBASE + 0x0180)
#define VIPC610_IDC      (void *) (VIPC610_IOBASE + 0x0280)
55 #define VIPC610_IDD      (void *) (VIPC610_IOBASE + 0x0380)

```

SUBSTITUTE SHEET

- 94 -

```
        /* VME medium/large memory space */  
#define VIPC610_MBASE      (void *) (VIPC610_IOBASE << 8)  
5 #endif
```

SUBSTITUTE SHEET

- 95 -

CLAIMS:

1. A robot comprising a body, and two legs without active joints, said legs being mounted to said body such that the legs can be independently translated and rotated with respect to said body.
5
2. A robot as claimed in claim 1, wherein said legs each include at least one foot.
3. A robot as claimed in claim 2, wherein the feet are adapted to retract and extend relative to the respective legs to engage and disengage a surface.
10
4. A robot as claimed in claim 3, further including abutting means which extend from and set the attitude of said body, wherein said feet on engaging said surface are adapted to retract to ensure the ends of said abutting means abut said surface.
15
5. A robot as claimed in claim 4, wherein the ends of said abutting means are configured to move over said surface and the motion of the legs is such that, at any given time during movement of said robot over said surface, said at least one foot of at least one of the legs engages said surface and said ends substantially continuously abut said surface.
20
6. A robot as claimed in claim 5, wherein the centroid of the feet in a plan view plane is within an area defined by the abutting means.
- 25 7. A robot as claimed in claim 6, wherein said centroid remains at or near the centre of gravity of said robot.
8. A robot as claimed in claim 2, wherein said legs each include three of said feet which define a triangle in a plan view plane, the apex of said triangle being on one side of said body and the base of said triangle being on the opposite side of said body.
30
9. A robot as claimed in claim 8, wherein the two triangles defined by said feet are

SUBSTITUTE SHEET

- 96 -

inverted with respect to one another, and the apex of each triangle extends beyond the base of the other triangle when the bases of said triangles are parallel.

10. A robot as claimed in claim 6, wherein said abutting means are disposed at at least respective corner parts of said body.

11. A robot as claimed in claim 10, wherein the feet of each leg are connected by transverse strut means of the leg which connect to a shaft of the leg substantially perpendicular to said strut means.

10

12. A robot as claimed in claim 11, wherein the strut means of one leg is disposed below the strut means of the other leg.

13. A robot as claimed in claim 12, wherein said shaft is driven by motor means mounted on said body to effect translation and rotation of said leg, translation of said shaft being restricted to one plane relative to said body.

14. A robot as claimed in claim 13, wherein said motor means includes two motors for each shaft, and uniform movement of said motors effects translation of the shaft and differential movement of said motors effects rotation of the shaft.

15. A robot as claimed in claim 2, wherein said feet each include a surface engaging shoe having means for adhering to said surface

16. A robot as claimed in claim 15, wherein said shoe is pivotal within a body of the foot to facilitate tangential engagement of said shoe with said surface.

17. A robot as claimed in claim 16, wherein said shoe is pivotal about a point below said surface.

30

18. A robot as claimed in claim 16, wherein said foot includes means for resetting the orientation of said shoe when retracted.

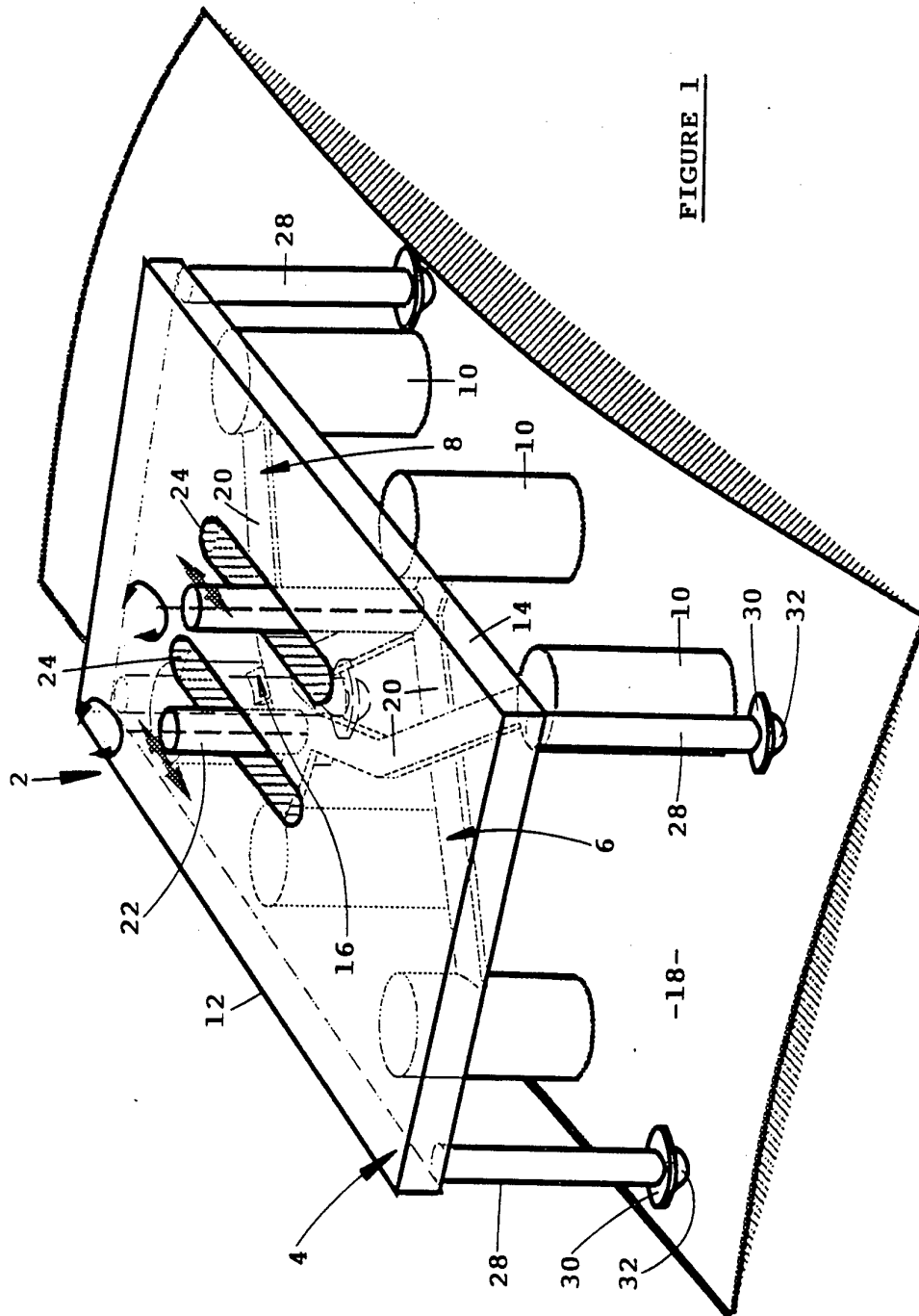
SUBSTITUTE SHEET

- 97 -

19. A robot as claimed in claim 15, wherein said adhering means comprises an electromagnet.
20. A robot as claimed in claim 15, wherein said feet include ram means for retracting
5 and extending said shoes of said feet.
21. A robot as claimed in claim 3, including means for confirming when said feet engage said surface.
- 10 22. A robot as claimed in claim 1, including means for providing position information to said robot, and means for controlling said legs in response to said information.
23. A robot as claimed in claim 22, wherein said information providing means comprises means for recognising a feature of the surface and determining said position
15 information for said feature relative to said robot, so as to cause said robot to track said feature.
24. A robot as claimed in claim 23, wherein the controlling means includes a pair of motors for each leg which act on a shaft of the respective leg, wherein uniform control
20 of said motors effects translational movement of said shaft and differential control of said motors effects rotational movement of said shaft relative to said body.
25. A robot as claimed in claim 24, including a tool mounted on said body and which is moveable relative to said body.
25
26. A robot as claimed in claim 25, wherein said controlling means further controls movements of said tool relative to said body in response to the position information.
27. A robot as claimed in claim 24, wherein said position information includes a
30 feature point representative of said feature, a tracking point, towards which said robot is to be moved, determined from the feature point, and a directional point being the location of said tracking point after said robot is moved.

- 98 -

28. A robot as claimed in claim 27, wherein said controlling means includes means for generating control signals for said motors to move said robot to effect transition from said tracking point to said directional point.
- 5 29. A robot as claimed in claim 28, wherein said tracking point moves along an axis between a sensor detection line and a tool movement axis to keep said feature in view of a sensor and to align tracking with the position of said tool.
30. A robot as claimed in claim 1, having means connected thereto means for
10 processing position information and means, responsive to said processing means, for controlling said legs.
31. A robot as claimed in claim 30, wherein the controlling means includes a pair of
15 motors for each leg which act on a shaft of the respective leg, wherein uniform control of said motors effects translational movement of said shaft and differential control of said motors effects rotational movement of said shaft relative to said body.
32. A robot as claimed in claim 31, wherein said controlling means further controls
20 means for retracting and extending said feet and means for adhering said feet to said surface.
33. A robot as claimed in claim 31, wherein said position information includes a
target point towards which said robot is to be moved at a selected speed, and said
processing means generates control signals for said controlling means to move said robot
25 towards said target point.



2/18

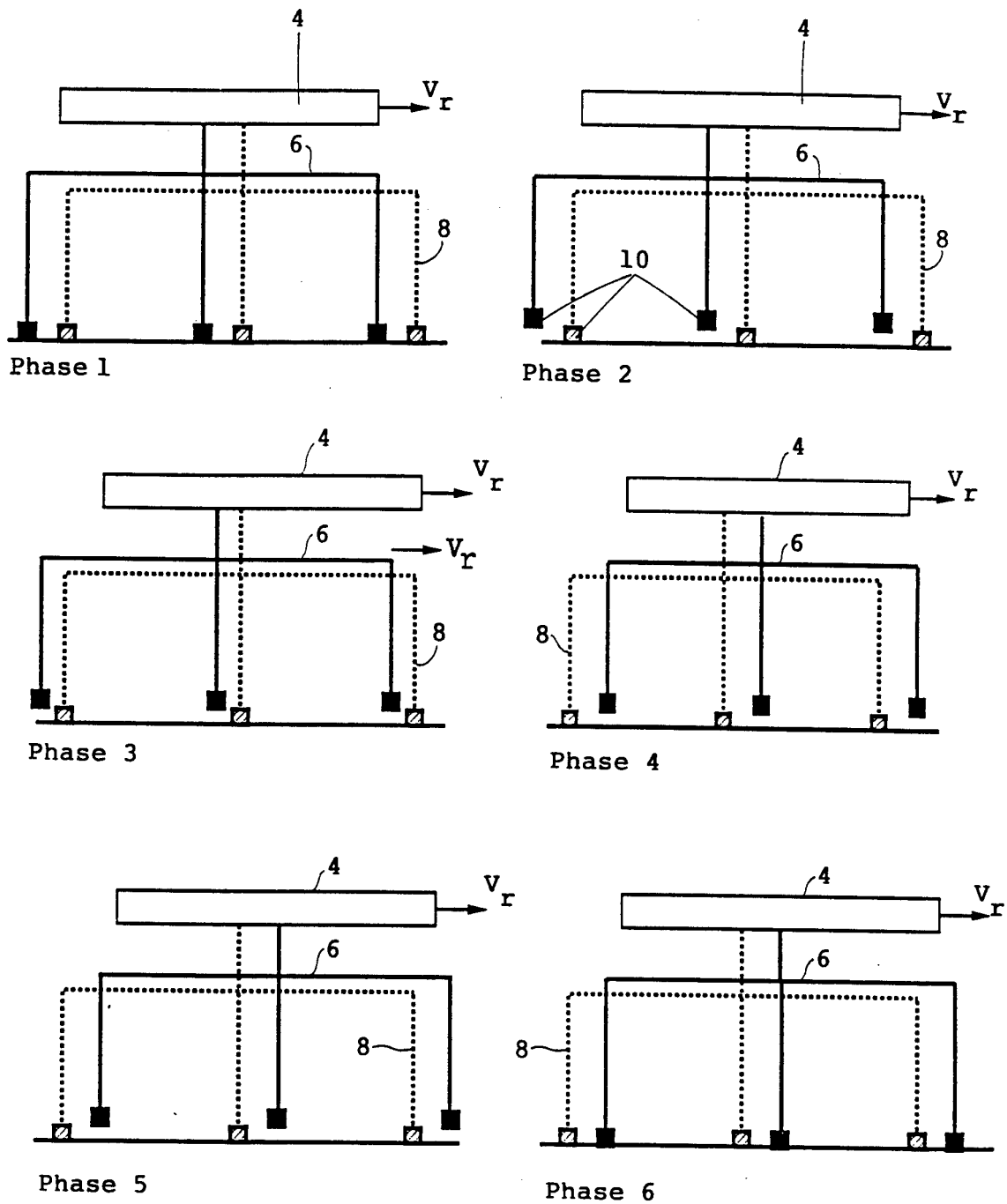
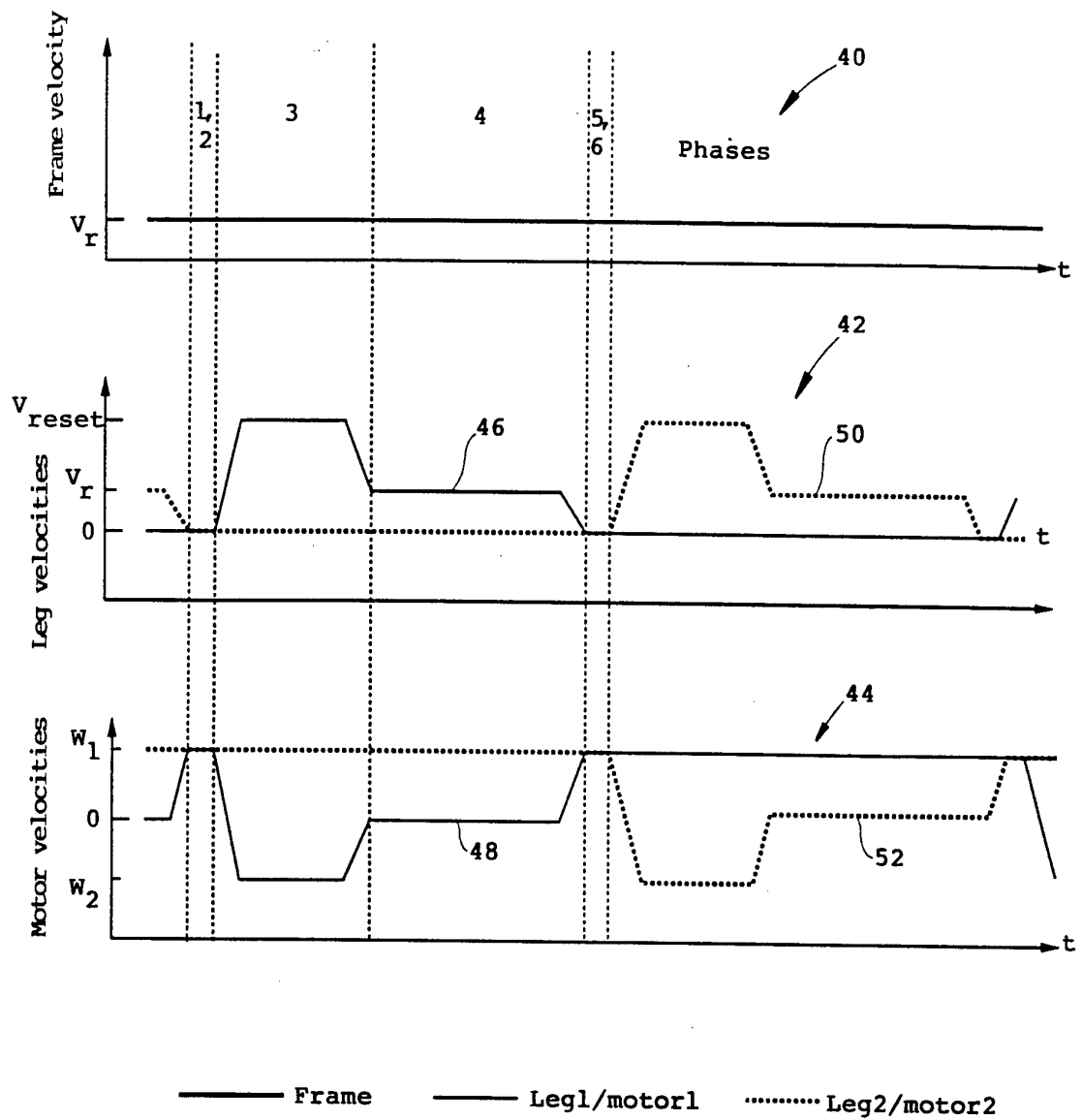


FIGURE 2

SUBSTITUTE SHEET

**FIGURE 3**

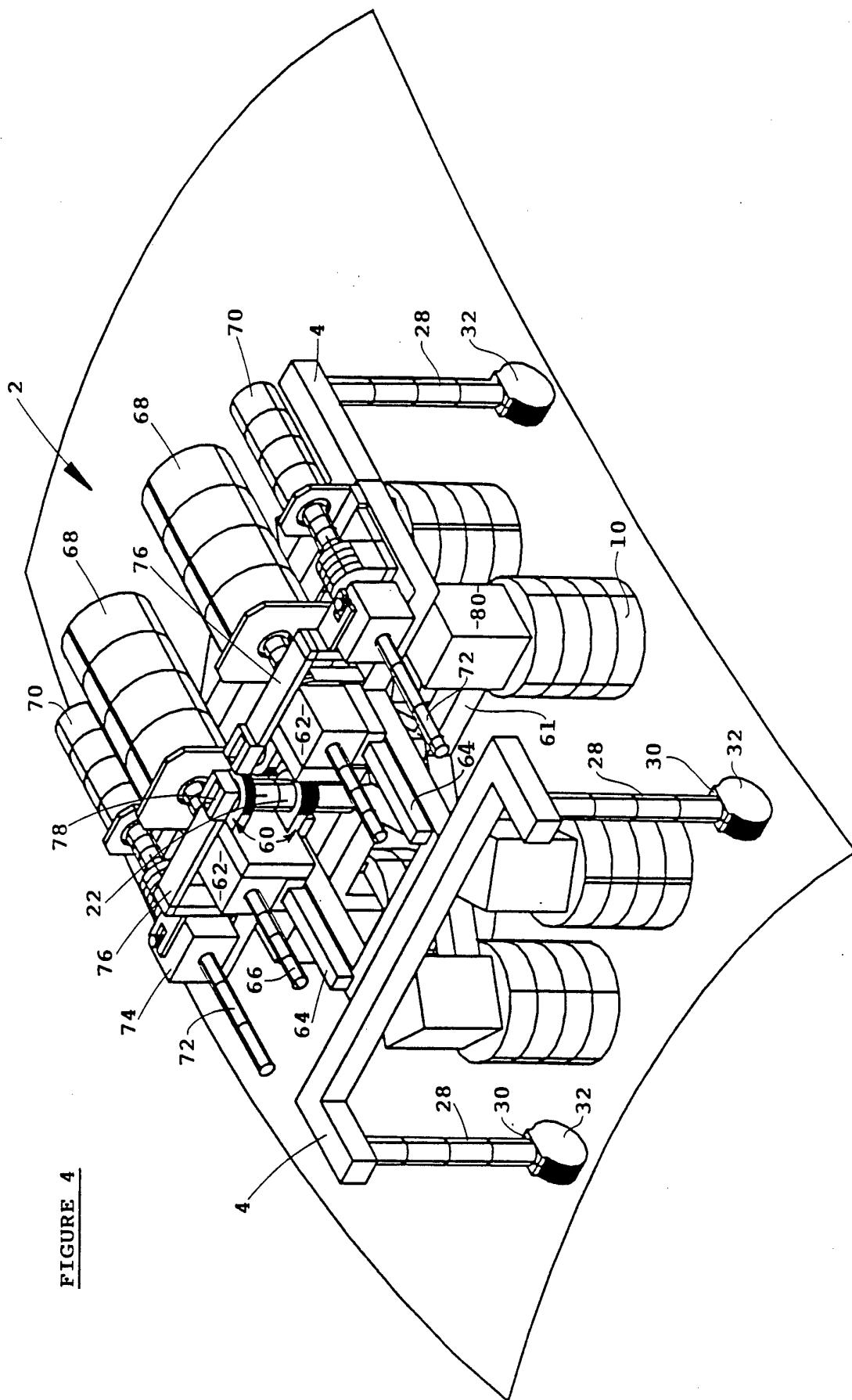


FIGURE 4

FIGURE 5A

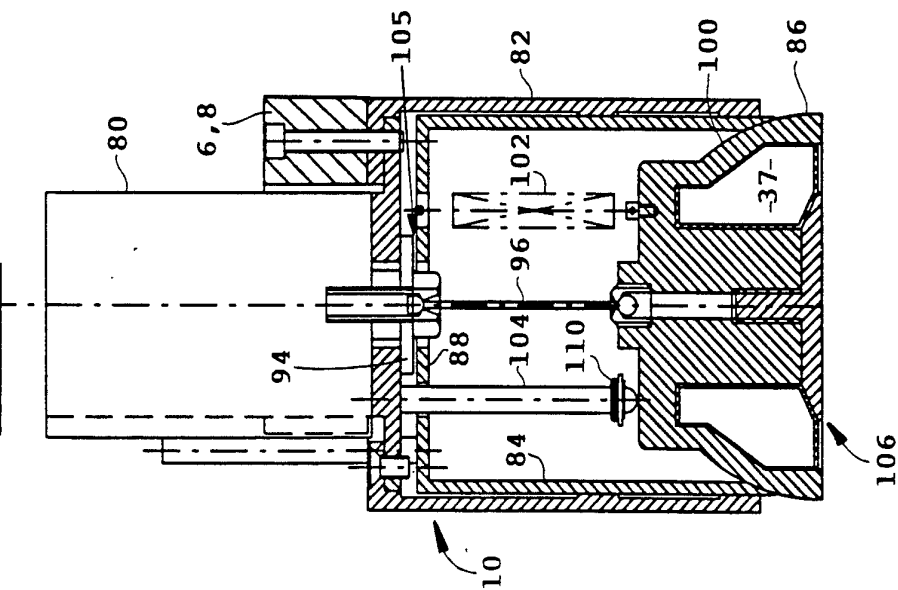


FIGURE 5B

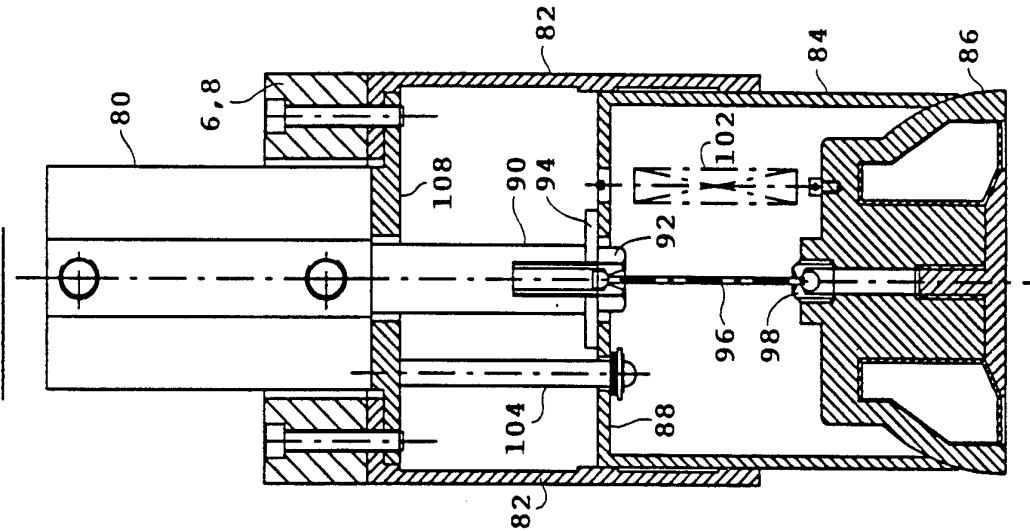
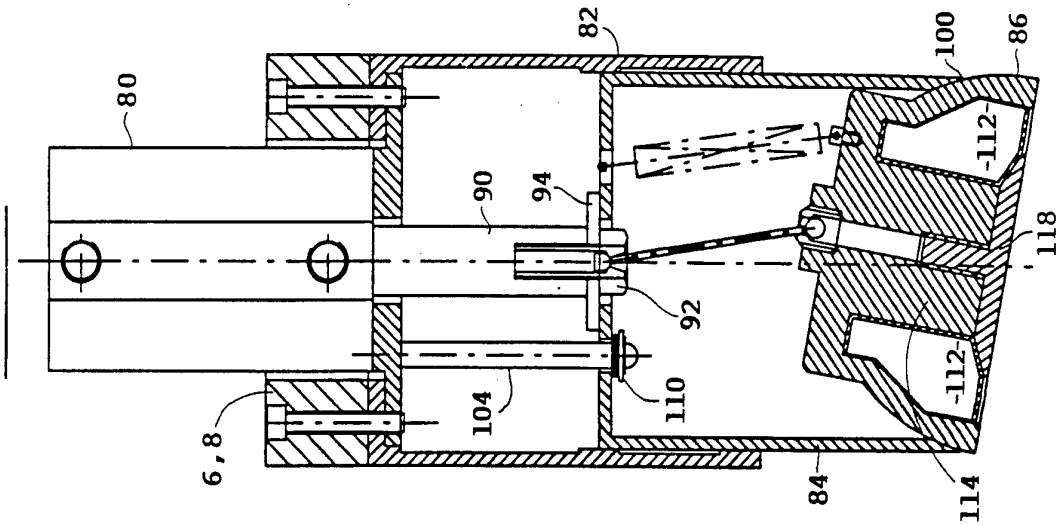


FIGURE 5C



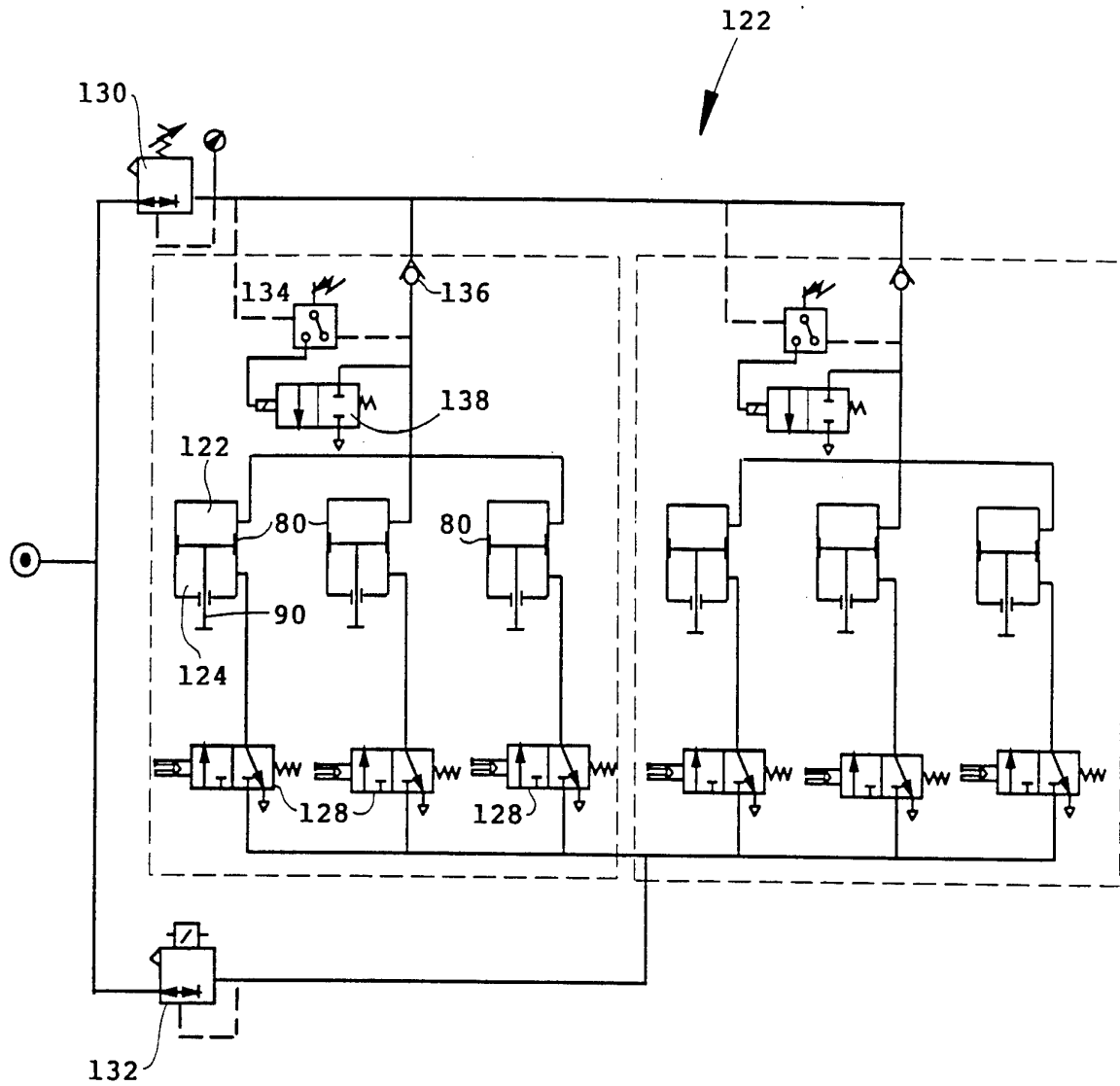


FIGURE 6

7/18

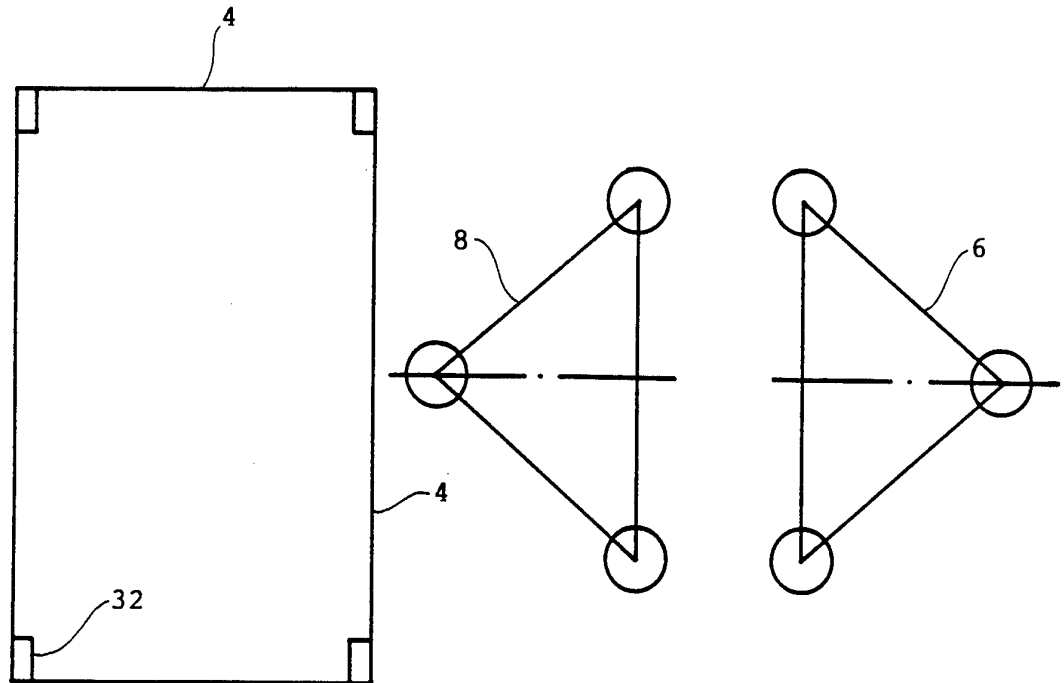


FIGURE 7

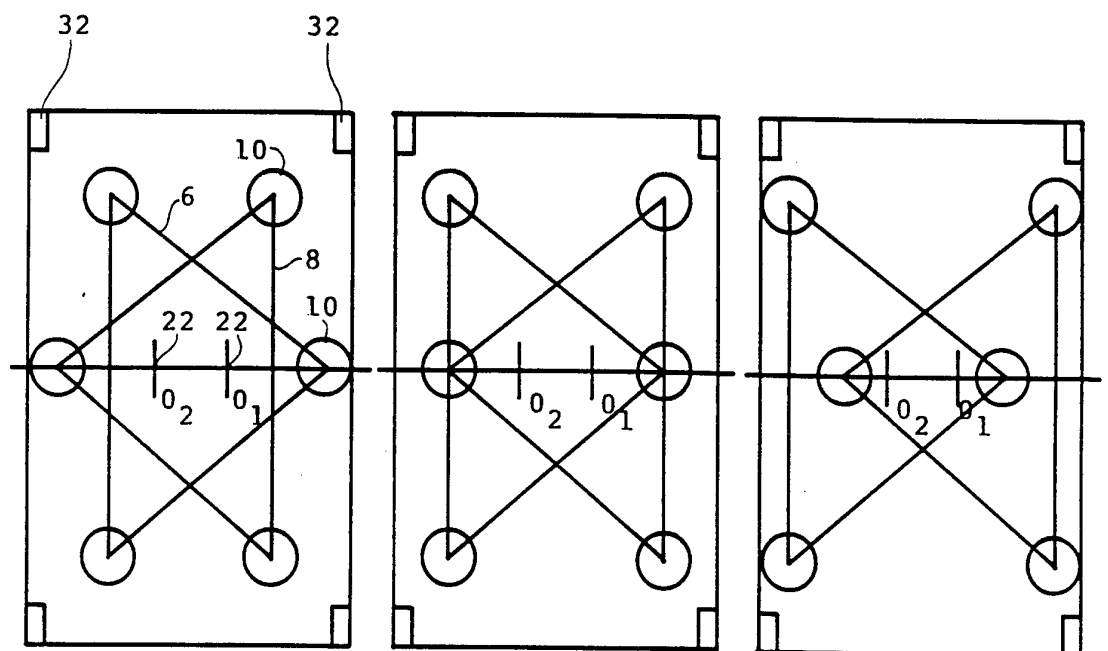


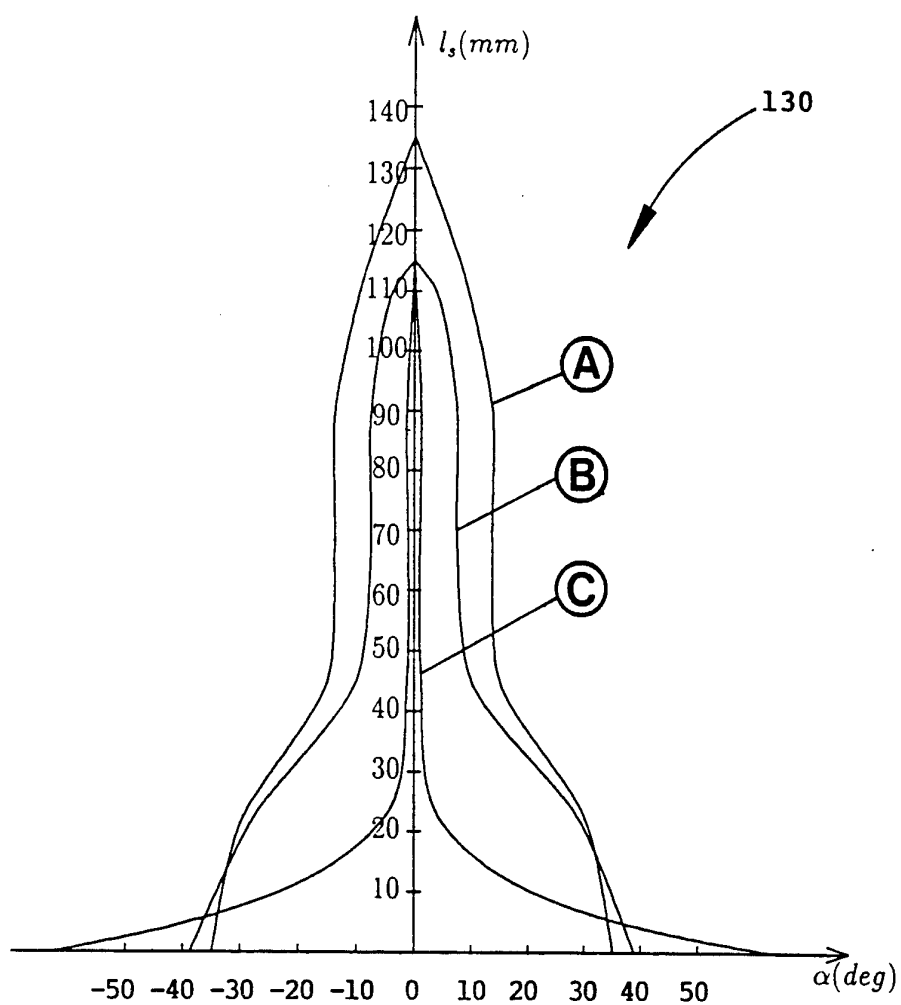
FIGURE 8A

FIGURE 8B

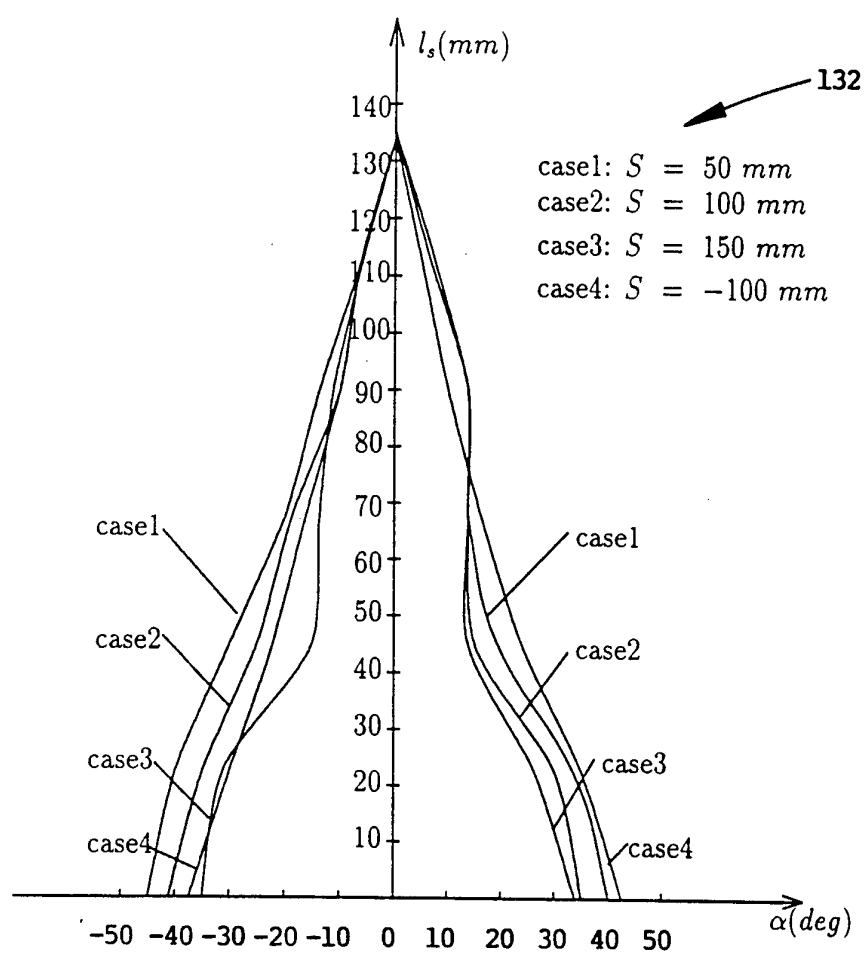
FIGURE 8C

SUBSTITUTE SHEET

8/18

FIGURE 9**SUBSTITUTE SHEET**

9/18

FIGURE 10

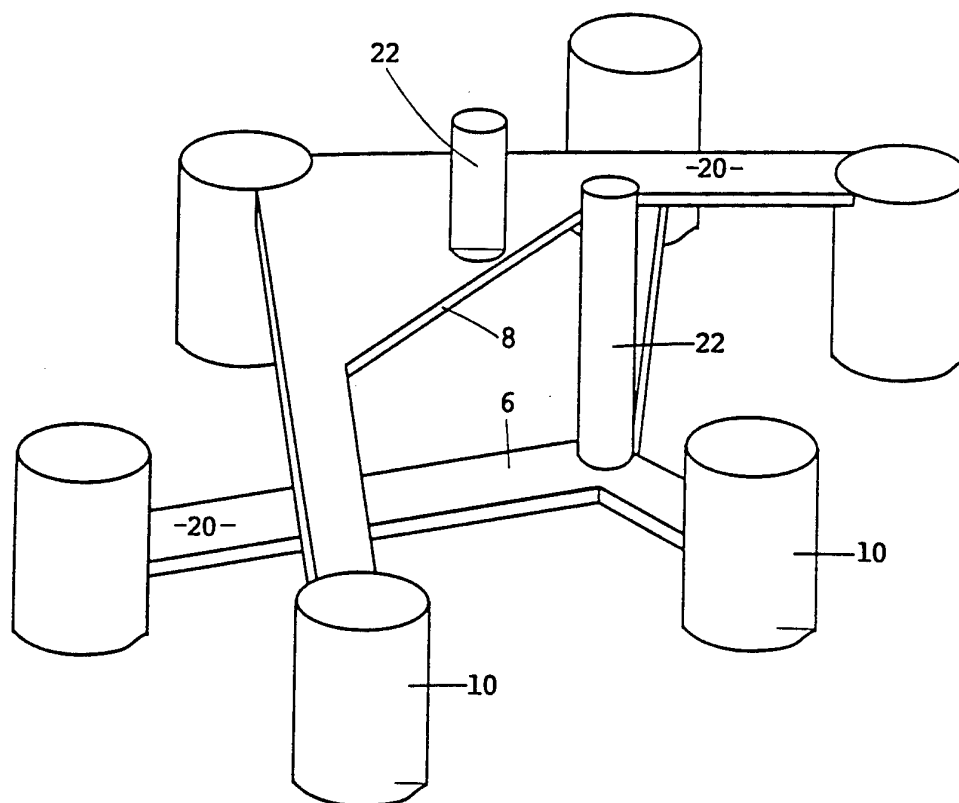
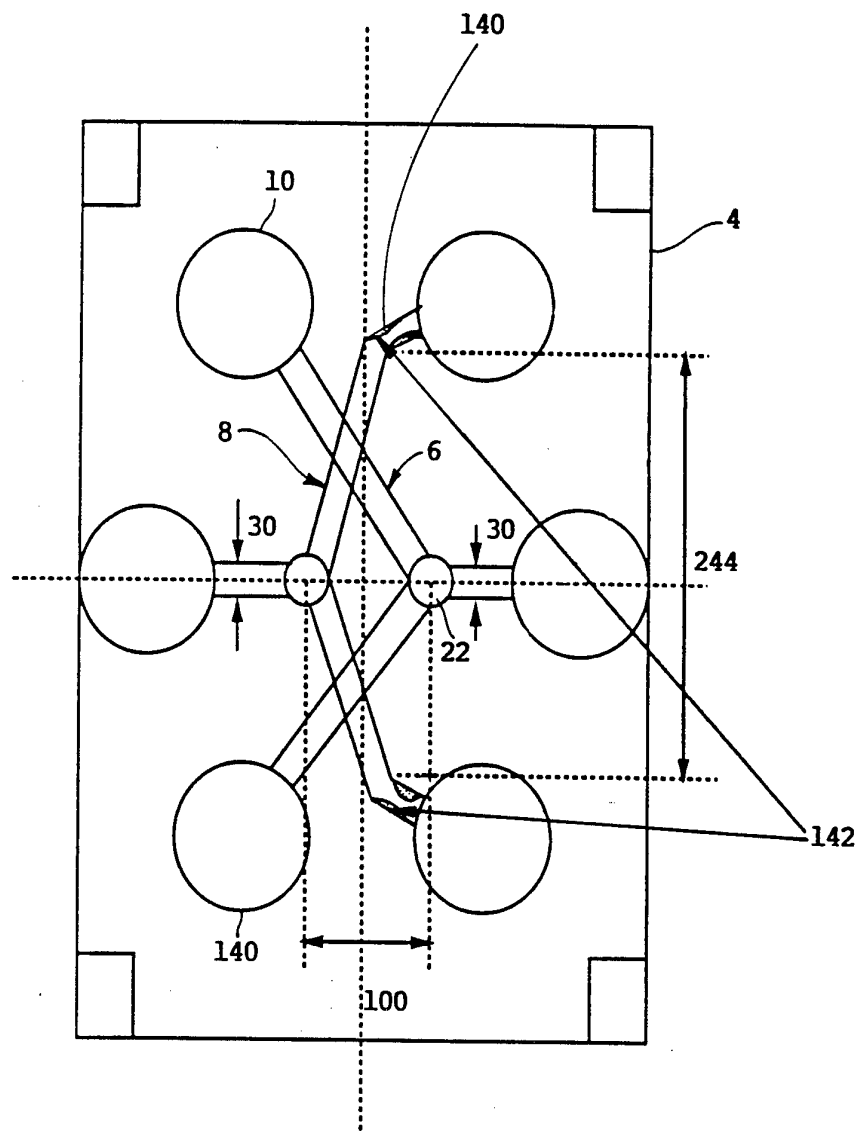


FIGURE 11

11/18

FIGURE 12**SUBSTITUTE SHEET**

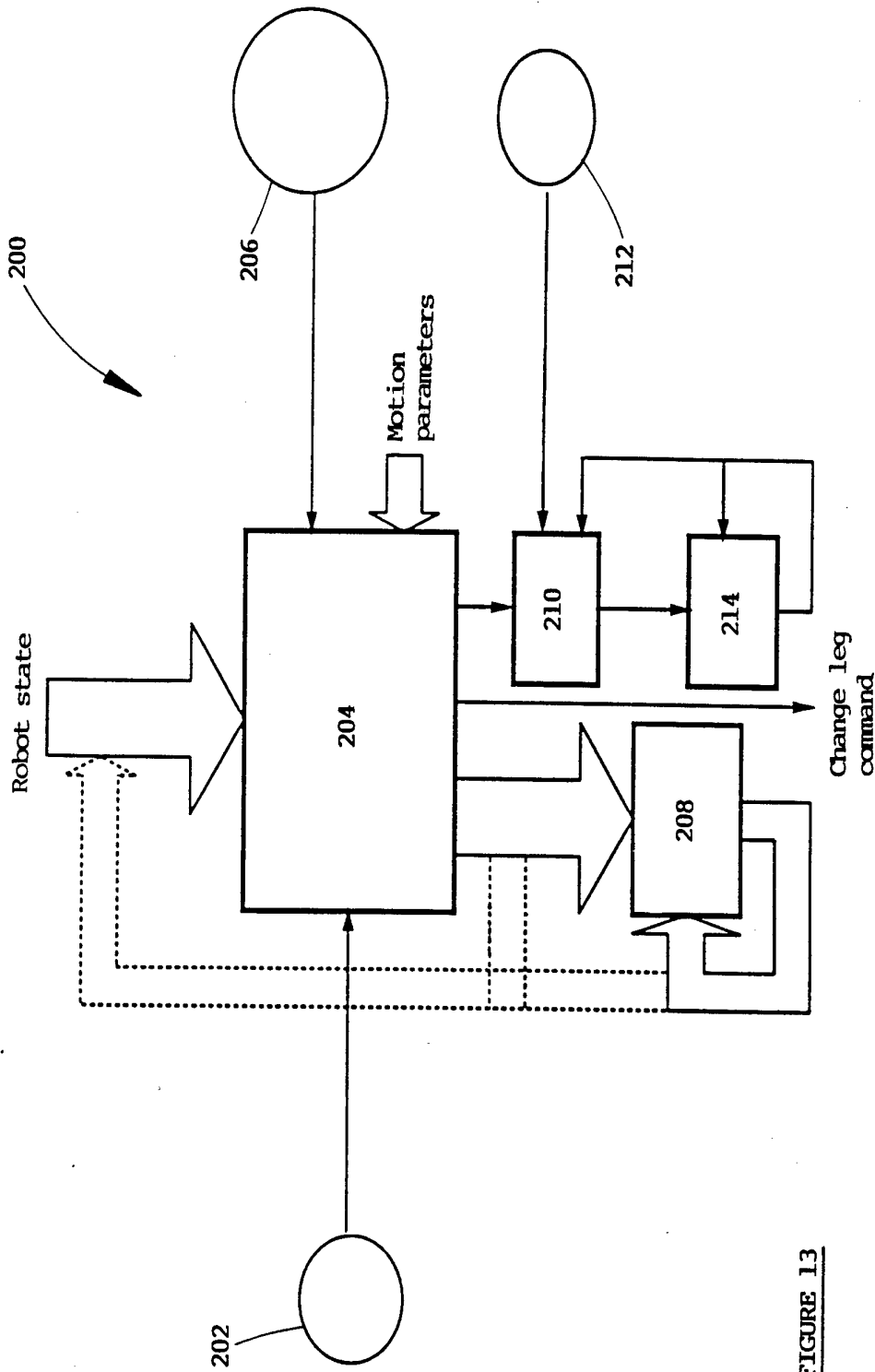


FIGURE 13

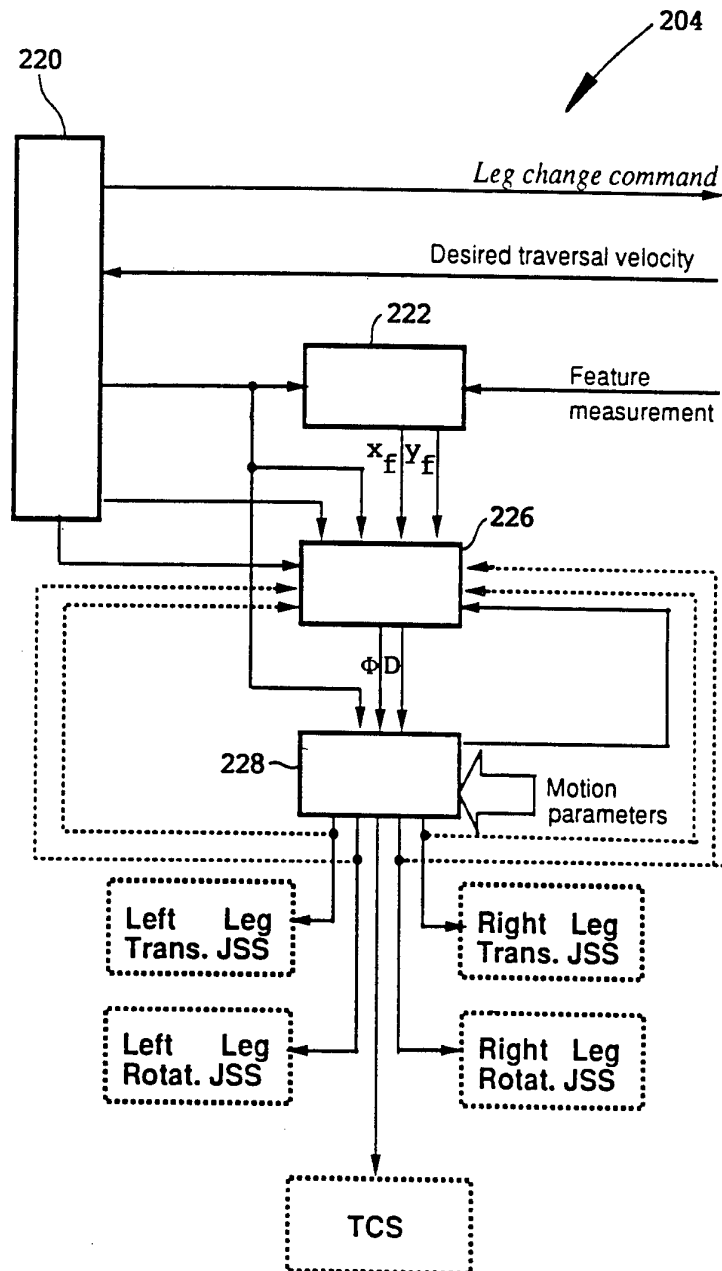
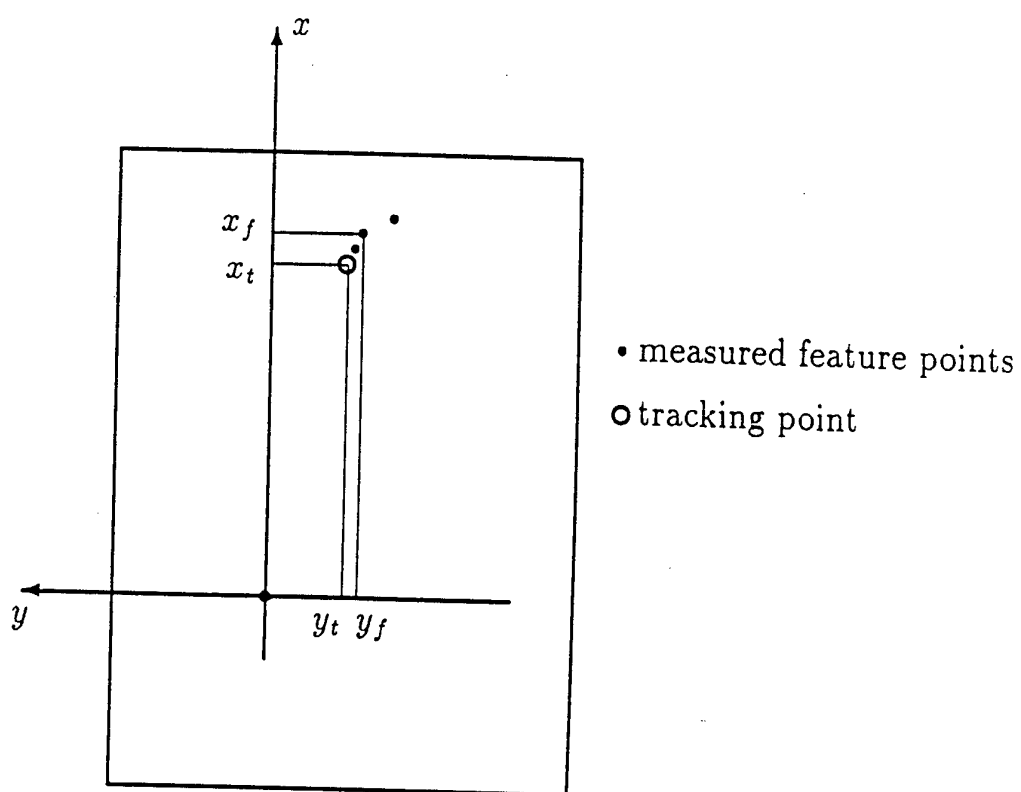


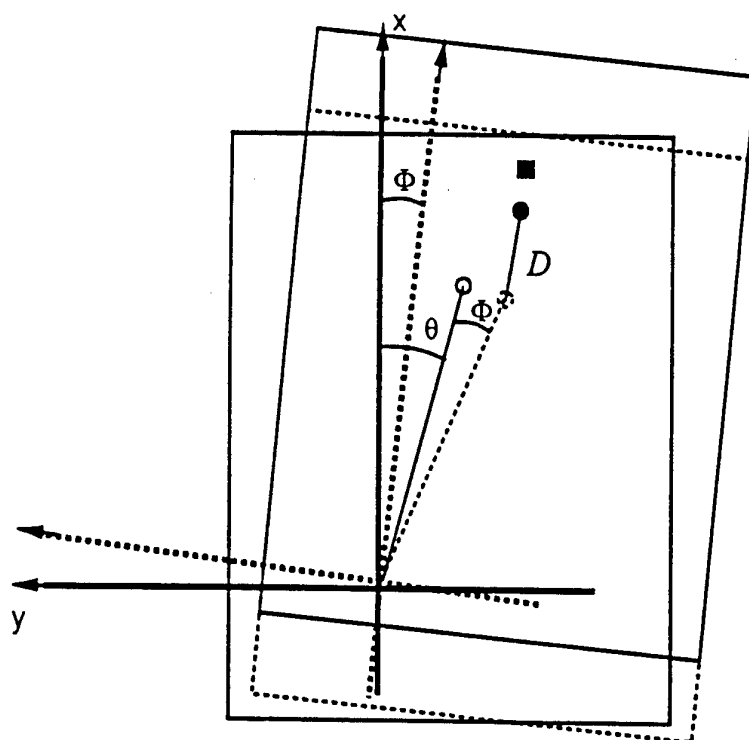
FIGURE 14

SUBSTITUTE SHEET

14/18

FIGURE 15**SUBSTITUTE SHEET**

15/18

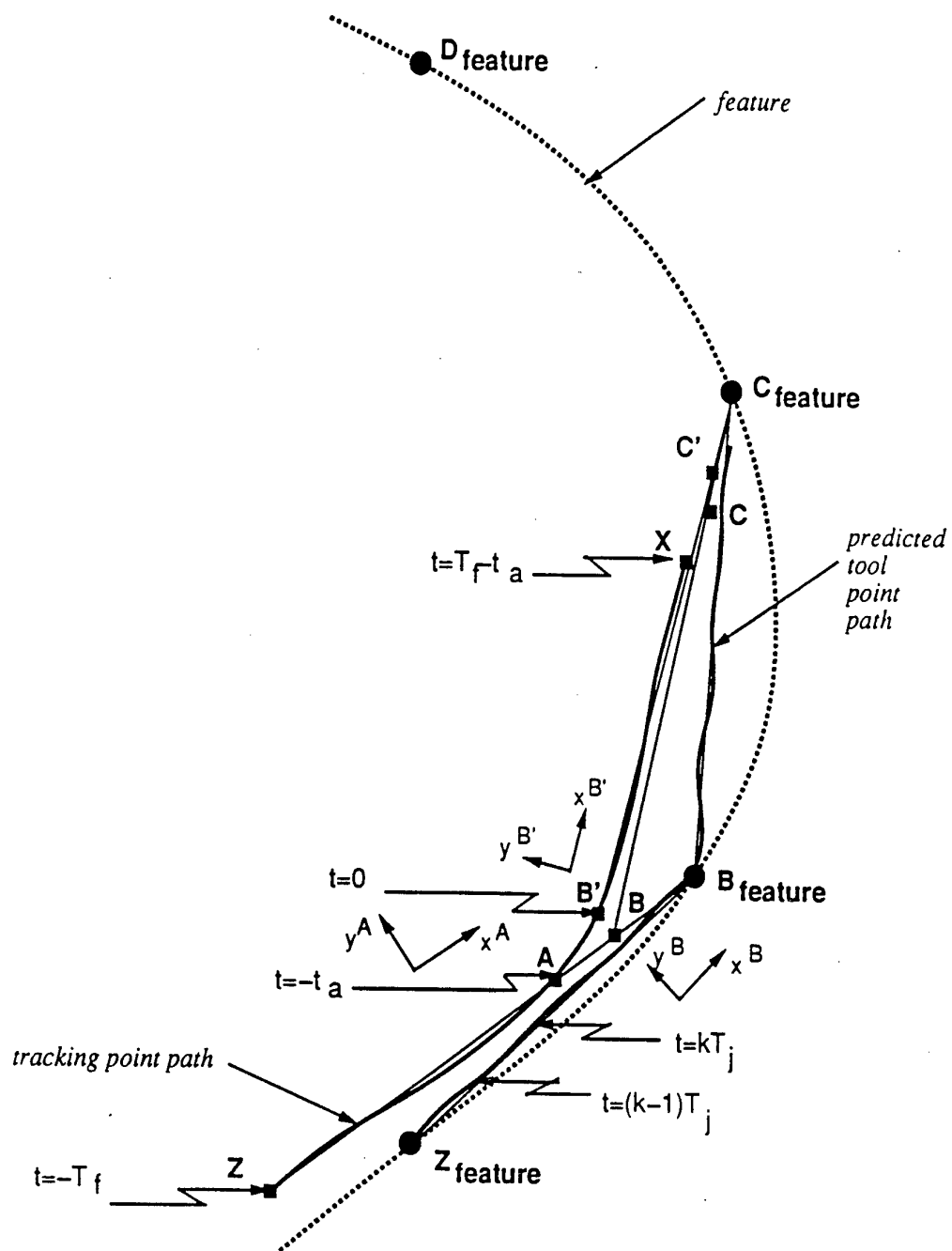


- directional point (x_d, y_d)
- tracking point (x_t, y_t)
- ⊙ tracking point after rotation
- tracking point after rotation and translation

FIGURE 16

SUBSTITUTE SHEET

16/18

FIGURE 17

17/18

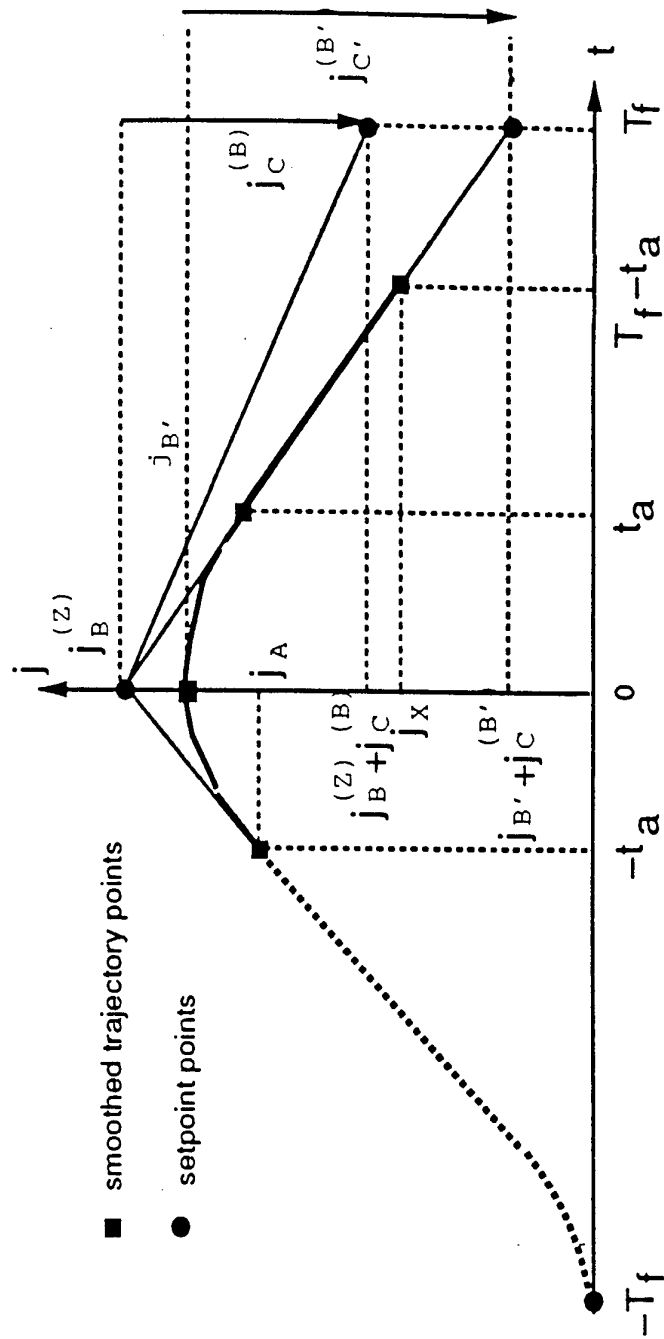


FIGURE 18

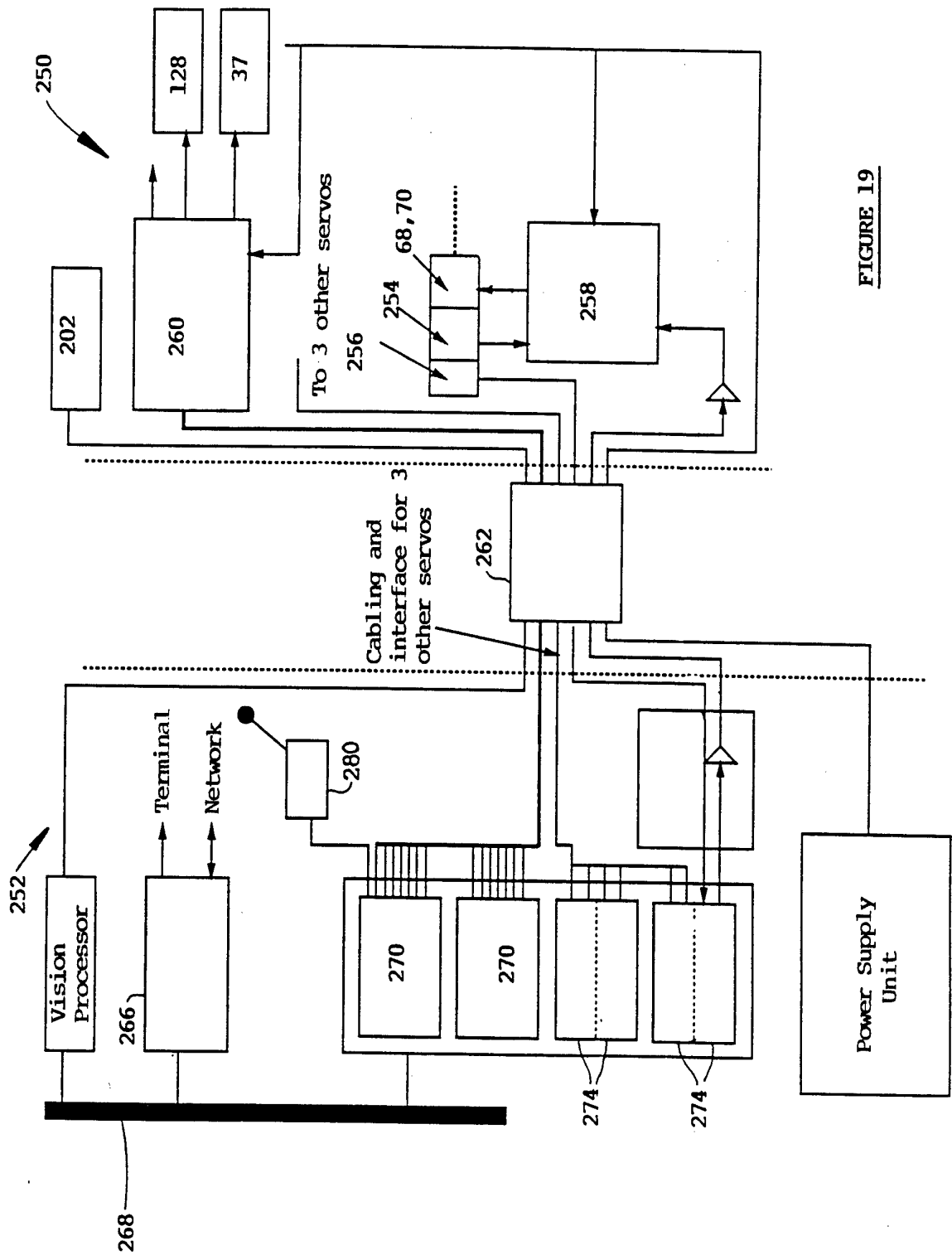
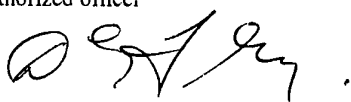


FIGURE 19

INTERNATIONAL SEARCH REPORT

International application No.

PCT/AU 93/00481

A. CLASSIFICATION OF SUBJECT MATTER Int. Cl. ⁵ B25J 11/00, 18/02, B23Q 16/00 According to International Patent Classification (IPC) or to both national classification and IPC				
B. FIELDS SEARCHED Minimum documentation searched (classification system followed by classification symbols) IPC : B25J 11/00, 18/02, 18/04, B23Q 16/00, G05D 1/10 Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched AU : IPC as above Robotics Research Volumes Published by MIT Press Electronic data base consulted during the international search (name of data base, and where practicable, search terms used)				
C. DOCUMENTS CONSIDERED TO BE RELEVANT				
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to Claim No.		
X A	US,A, 4832563 (MASSMANN) 23 May 1989 (23.05.89) Figure 1 Entire document	1-7 8-33		
Y A	US,A, 5094311 (AKEEL) 10 March 1992 (10.03.92) Figures 2,3 Entire document	1-11 12-33		
Y A	US,A, 4565487 (KROCZYNSKI) 21 January 1986 (21.01.86) Figures 4,5 Entire document	1-11 12-33		
☒ Further documents are listed in the continuation of Box C. ☒ See patent family annex.				
<table style="width: 100%; border: none;"> <tr> <td style="width: 50%; vertical-align: top;"> * Special categories of cited documents : "A" document defining the general state of the art which is not considered to be of particular relevance "E" earlier document but published on or after the international filing date "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified) "O" document referring to an oral disclosure, use, exhibition or other means "P" document published prior to the international filing date but later than the priority date claimed </td> <td style="width: 50%; vertical-align: top;"> "T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention "X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone "Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art "&" document member of the same patent family </td> </tr> </table>			* Special categories of cited documents : "A" document defining the general state of the art which is not considered to be of particular relevance "E" earlier document but published on or after the international filing date "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified) "O" document referring to an oral disclosure, use, exhibition or other means "P" document published prior to the international filing date but later than the priority date claimed	"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention "X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone "Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art "&" document member of the same patent family
* Special categories of cited documents : "A" document defining the general state of the art which is not considered to be of particular relevance "E" earlier document but published on or after the international filing date "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified) "O" document referring to an oral disclosure, use, exhibition or other means "P" document published prior to the international filing date but later than the priority date claimed	"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention "X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone "Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art "&" document member of the same patent family			
Date of the actual completion of the international search 13 December 1993 (13.12.93)		Date of mailing of the international search report 20 DEC 1993 (20.12.93)		
Name and mailing address of the ISA/AU AUSTRALIAN INDUSTRIAL PROPERTY ORGANISATION PO BOX 200 WODEN ACT 2606 AUSTRALIA Facsimile No. 06 2853929		Authorized officer  D.G. FRY Telephone No. (06) 2832130		

INTERNATIONAL SEARCH REPORT

International application No.
PCT/AU 93/00481

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT		
Category*	Citation of document, with indication, where appropriate of the relevant passages	Relevant to Claim No.
X	US,A, 4248103 (HALSALL) 3 February 1981 (03.02.81)	1-7
A	Col 1 lines 7-66, Figure 1	8-33
	Entire document	
	Derwent Abstract Accession No. E 7056 K/14, Class P62, SU,A, 932017 (KUDINTSEV) 30 May 1982 (30.05.82)	
Y	Abstract	1-13
A	Abstract	14-33
	Derwent Abstract Accession No. 90-170264/22, Class P62, SU,A, 1523337 (NOVOS POLY) 23 November 1989 (23.11.89)	
Y	Abstract	1-7
A	Abstract	8-33
	Derwent Abstract Accession No. 90-020983/03, Class P62, SU,A, 1472252 (AZERB POLY) 15 April 1989 (15.04.89)	
A	Abstract	1-33
	Derwent Abstract Accession No. 89-376481/51, Class P62, SU,A, 1463467 (KAZAN AVIATION INST) 7 March 1989 (07.03.89)	
A	Abstract	1-33
	Derwent Abstract Accession No. 86-040903/06, Class P62, SU,A, 1142271 (NOVOS ELECTROTECH INST) 28 February 1985 (28.02.85)	
A	Abstract	1-7

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/AU 93/00481

This Annex lists the known "A" publication level patent family members relating to the patent documents cited in the above-mentioned international search report. The Australian Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

Patent Document Cited in Search Report		Patent Family Member					
US	4248103	CA	1125812	EP	1163	JP	54049757
US	4565487	CA	1166670	EP	74286	FR	2512410
US	4832563	DE	3626610	EP	255612		
END OF ANNEX							