An Autonomous system for the locomotion of a Hexapod Exploration Robot

Pablo Muñoz-Martínez†, María D. R-Moreno†, Javier Gómez-Elvira§,
Julio J. Romeral-Planelló§ and Sara Navarro-López§
†Departamento de Automática, Universidad de Alcalá
28805 Alcalá de Henares (Madrid), Spain
Email: {pmunoz, mdolores}@aut.uah.es
§ Centro de Astrobiología (CAB)
28850 Torrejón de Ardoz (Madrid), Spain
Email: {gomezj,romeralpj, navarrols}@inta.es

***Abstract*—Ptinto is a hexapod robot designed to keep the equilibrium when moving around rocky and cumbersome areas during the exploration of the Tinto river in Huelva (Spain).**

We have developed an integrated planning and scheduling system called PIPSS to control the locomotion of the P-Tinto robot. PIPSS tries to make the better moves for the legs in order to keep the right balance and calculate the trajectory between two points. It exchanges information with an executor system that execute the plan, and in case there are some obstacles that Ptinto cannot avoid, a new trajectory will be re-calculated.

I. Introduction

The Tinto river crosses the South-West part of Spain. In its way to the Atlantic Ocean, it crosses a miner area rich in heavy metals. The water, by the metal effects, presents some special features that make it very similar to the Mars planet: is red, dense and with limited oxygen, what has made scientist think that there is no life. For this reason, this area has been used and studied by NASA researchers to test different components that would be sent to Mars. Recent studies have demonstrated that there exist microscopic organisms that can life in that environment. In order to study them, the Astrobiology Center (CAB) in Spain is developing a terrestrial robot called Ptinto to analyze shallow and difficult access areas in the Tinto river.

This paper represents a first step to make an autonomous control system for the locomotion of Ptinto robot. The main objective is to implement a control system for the locomotion of the robot using the PIPSS system [2]. With the plans generated by PIPSS, the control system is able to translate the orders to messages that the legs can understand and execute. The sequence of orders, and the interaction between PIPSS and the hardware of the robot is controlled by the system, designed to be able to trace a path between two or more points using the information generated. Also, the system needs to be able to avoid obstacles, if they are any, and to reach the desired goals.

The paper is structured as follows. First, a description of the Ptinto robot is provided. Next, section III presents the planning and scheduling system that is part of the control system. Then, in section IV the architecture for the Ptinto robot is described. After, section V briefly describes the simulator built to test the architecture. Finally, some conclusions and future research lines are outlined.

II. Ptinto Exploration Robot

The legged robots represent an alternative to the traditional wheels robots like the NASA Mars exploration rovers [7], [8].

It is proven that they were quite inefficient in rocky and cumbersome environments. Legged robots offer better mobility in difficult terrains and they are more capable to overcome obstacles [6]. The main features of the Pinto robot were presented in 2005 in the Universal Exposition that was held in Japan. The aim to build Ptinto was to send part of its technology in the first European expedition to Mars. Among the more relevant technologies that will be implemented, we can mention their sensors, with DNA chip, that should allow the scientists to precise if there are primitives bacteria or other primary organisms in the Mars land.

The Ptinto exploration robot is composed of six legs (hexapod) attached to a hexagonal chassis. Figure 1 shows the current state of Ptinto.

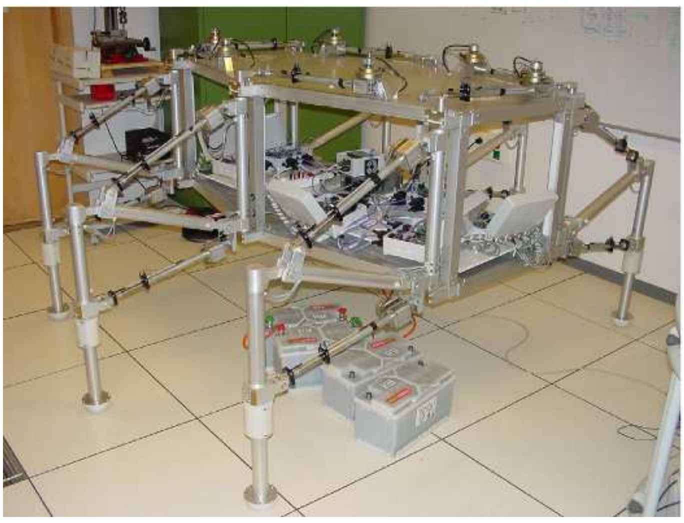

Fig. 1. Current state of the Ptinto exploration robot

This robot has one mini PC with Debian embedded operating system for the main program control, connected with all the pods with a CAN bus. Also, each pod has a microprocessor with its necessary electronic components and low level programming to provide an interface with simple control commands. Currently, the only sensors the robot has are contact detecting obstacles for each pod.

Each pod has three degrees of freedom for its mobility, each one controlled by one linear actuator. The linear actuators are connected to the microprocessor that takes the control of the leg and of a group of photodiodes to prevent an excess in the opening angle of the leg. The microprocessor is also connected to the CAN bus to read the orders that the main control program sends through the bus. These orders are a small group of functions defined in the microprocessor, and with them, we can control the entire leg. These functions include move the leg up or down, go forward or backward, etc. When the movement is complete, the legs send a message to the main control program via CAN bus with the status of each leg, that is, the degree of all the motors. A diagram of the allowed movement of the leg, the motors position in the leg and all the legs numbered from an upside vision of Ptinto is shown in figure 2.

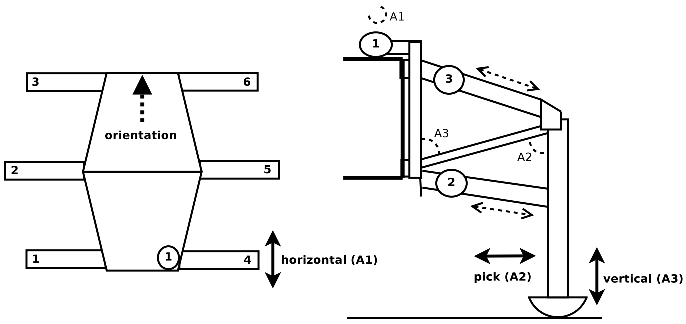

Fig. 2. Upside view of Ptinto and one of its legs

For obstacle detection, each leg has three extensiometric gauge bridges. These bridges send an electrical signal when they come in contact with an object. Two of the bridges are in the axis of the leg to detect obstacles that are higher than the robot can avoid, and the last bridge is on the toehold of the leg. This last one is very useful to walk over soft and slippery surfaces, because it indicates the degree of support of the surface.

All legs are controlled by a PC104. It is a small, lightweight and standard PC designed by the PC104 Embedded Consortium [1] for embedded applications. It is a full operative PC with a Linux kernel, easily upgradable with its hardware module design. The version installed is enlarged with a CAN bus controller for the communication with the legs. Also, it can incorporate modules like GPS or wireless interfaces. Software support is the same as other Debian distributions; in the PC104 we can install any needed application and we can develop the control program in the programming language that we need, before installing the compiler. For the maintenance of the PC it has a serial and an Ethernet port.

Ptinto is designed to transport near 50 kg of scientific load on his chassis. It has 168 cm long and between 57 (front and rear) and 90 (center) cm wide. The scientific load is under development. The initial idea is to install a microscope mounted on a movil arm, DNA chips to evaluate the presence of microscopic life, a spectrometer, some physicochemical sensors and a system to collect samples of the terrain. Another interest issue is to install a camera system to allow remote control operations. All of these systems will be connected to the main PC via CAN bus or an Ethernet link, depending on the needs.

III. The planner and the scheduler: PIPSS

One of the main pieces for the Ptinto control system is PIPSS (Parallel Integrated Planning and Scheduling System) [2]. It is based on the IPSS (Integrated Planning and Scheduling System) [3] philosophy. This philosophy involves adding the particularities of a planner and a scheduler in a single system. Therefore, the planning can work with decision trees pruned by the scheduling system. That feature means that the integrated system can operate with time and constraints within the planning reasoning.

PIPSS implements this philosophy, integrating HPP (Heuristic Progressive Planner) [4] like the planner component, and the O-OSCAR (Object-Oriented Scheduling ARchitecture) [5] main algorithms as the scheduler. HPP is a heuristic progressive PDDL planner based on the FF planner [9]. But HPP introduces improvements in the operators instantiation. It includes a new module that is able to exclude irrelevant domain-dependent operators for the planning process. The main algorithm used by O-OSCAR is ISES (Iterative Sampling Earliest Solutions) [10]. Basically, ISES is a simple optimization algorithm that iterates another algorithm called ESA (Earliest Start Algorithm), which is in charge of returning time and resources consistent solutions. ESA solves temporal restrictions using a temporal network and avoids conflicts due to resources by imposing additional precedence relations between pairs of activities that are responsible for such conflicts. ISES just asks ESA for several solutions to try to find one with a better makespan.

The novelty of PIPSS is that it defines an interface for each of the most important components of the planning and scheduling integrated search, which are: planning algorithm, scheduling algorithm and planning and scheduling integrated search scheme. This allows the cooperation of different components of these kinds, which can work together regardless of their particularities. Various standard patterns have been defined for each of these components that all algorithms must compulsory comply with. This way, it is possible to connect any planning and scheduling algorithms with any kind of integration scheme without the need to modify or enlarge other parts of the system. The PIPSS language is based on PDDL (Planning Domain Definition Language) [11] but extended to reason about resources, temporal windows and deadlines. Finally, PIPSS is designed to be parallel, this implies two parallel degrees: one is the parallel execution of the planner and the scheduler, and the other is given by HPP, which realizes parallels searches into the states tree.

PIPSS integrated search is the same that IPSS uses to integrate planning and scheduling. Its work scheme can be seen in figure 3. To sum up, the planner generates a total order plan that is then processed by the deordering algorithm, which passes the obtained partial order plan to the scheduler, composed by a temporal network and a network for resources handling. If the plan is feasible and reaches the goals of the problem, it is returned as a solution. If it is not feasible, the planner is asked to backtrack.

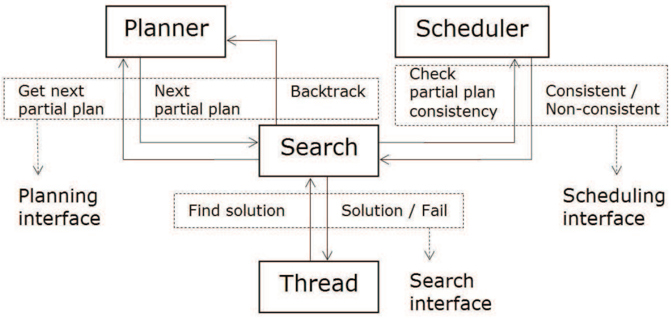

Fig. 3. PIPSS architecture

IV. The Proposed Architecture

The Ptinto autonomous control architecture is a traditional 3 tiers or 3T architecture. The lowest tier constitutes the functional layer. The middle tier is an executive that executes commands to achieve the goals. The top tier is the planning tier that uses the AI Planning and Scheduling system PIPSS.

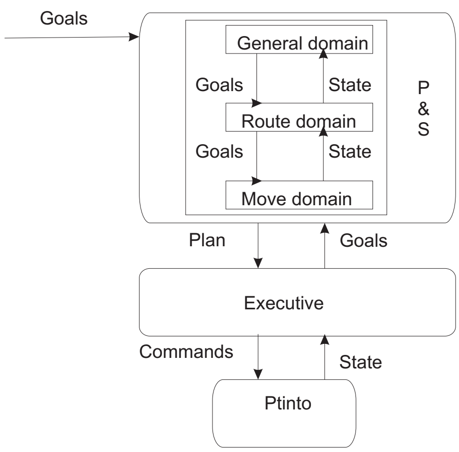

Fig. 4. Architecture overview

Due to the complexity of calculating the path and the legs that have to be moved to keep the equilibrium, we have chosen to break down the planning domain into layers, each one operating at a different level of granularity in a similar way as done in [6]. Figure 4 shows the different layers that compose the architecture and the P&S tier break down into its different layers.

One layer will calculate the trajectory (to avoid detected obstacles) and another one will optimize the legs movement (based on time and costs). Then, the top one will generate a top level plan that will serve as a baseline for the other two. When more functionality is added to Ptinto, it is likey that we need to add some more layers into the planner domain.

The path calculus consists of a route among two or more waypoints. These waypoints are represented like pairs of coordinates in the Cartesian axis. The reason to do that is because the robot does not yet include a position system. So the control system needs to control the position by a reference system. This implies a mathematical base to calculate the rotation angle and the distance between the actual position and the next reach point as it is shown in figure 5.

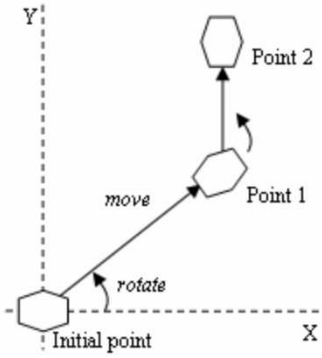

Fig. 5. Move of Ptinto robot

The locomotion control of the robot is upside down. First, the planner generates a sequence of orders to trace the route. This route is represented by a sequence of inter-connected points. To reach each waypoint, the robot needs to perform two moves: rotate to orient itself to the point direction; and go forward until achieve that waypoint. These moves are calculated also by the planner that generates a plan of orders to move all the legs as needed. When the first point is reached, the robot continues reaching the next point, and it does that until it arrives to the final point designated in the route. If in one of the moves the robot cannot achieve the planned point, it implies that there is an obstacle between the actual position and the next point. To avoid the obstacle, we consider a new intermediate point between the previous waypoint and the point the robot needs to achieve as shown in figure 6. Next subsections describe in detail each layer.

A. *Upper tier: planning and scheduling*

The top layer is represented by the planning and the scheduler reasoner PIPSS. It receives the goals that have to be achieved, and it generates a plan composed by the sub-plans solved by each of the 3 layers that the domain is subdivided in. Each layer will be next explain in detail.

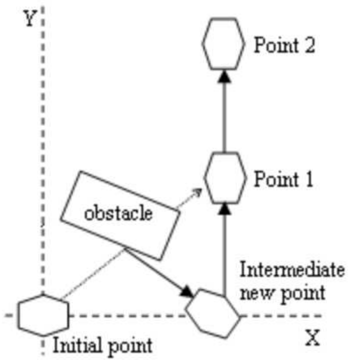

Fig. 6. Move after obstacle detection

1) General domain: This layer will generate a top level plan that will serve as a baseline for the other two. The plan generated right now is very simple, since there is still a lot of information that could be obtained from other sensors or systems that will be added to Ptinto in the future and are not yet implemented. But the way it is designed will allow us to easily introduce more functionality.

2) Route domain: As mentioned before, the plan for the route domain consists of a set of waypoints between two points (initial and end) that should be inter-connected. The goal of the route planner is to generate a reasonable (it does not have to be optimal) route through the different points. The waypoints are represented by the Cartesian coordinate system due to the lack of a GPS. To represent each point uniquely in a plane, two numbers are used, usually called the x-coordinate or abscissa and the y-coordinate or ordinate of the point. From each waypoint, there are one or more waypoints that can be reached. It implies that is possible to go between two waypoints using different routes. The election of the route concerns to PIPSS based on the distante among the waypoints. We are not considering in this first approach the type of terrain because this would complicate the solution since some kind of terrain such as flat would be preferred to rocky.

For obstacle avoidance, the system generates two new waypoints (one for the actual position and other to avoid the obstacle) and the corresponding connections: one between the actual position and the new waypoint, and another connection to reach the destination before the obstacle detection and the new generated waypoint. To generate it, we estimate from the sensors in the legs how big the obstacle can be and the angle that the robot should rotate to avoid it. With this information the new waypoint is created.

3) Move domain: The plan for the movements generates a sequence of moves for each leg. Actually there are only three moves: go ahead, rotate to the left and rotate to the right a specific angle. It is also determined if it is just one leg that has to be moved or two at the same time.

A correct sequence of moves allows Ptinto to reach the next waypoint in the route plan. To do this, the system calculates the rotation needed and the distance between the actual position of the robot and the next waypoint. Each move is modelled as a succession of moves for all the legs. These moves are a representation of the possibilities defined in the hardware control such as move the leg ahead or backward.

B. Middle tier: executive

The middle tier represents the executive. It is the layer in charge of executing the plan given by PIPSS. It interprets the plan generated looking only the high level orders. For each plan it can read one order when it is requested and translates it to the hardware abstraction layer. It also analizes at each step if the command(s) executed produces the expected result. If not, it asks the upper layer to re-plan.

C. Lower tier: hardware control

This layer implements the interface to accept and execute the orders from the executive. It also provides the actual status to the control system and a mechanism to check the execution of the orders. It allows us to work with an abstraction of all the hardware of the Ptinto robot. This abstraction represents the legs and the CAN bus. For each hardware module, there is one software module that encapsulates the functionality and provides a high level function to work with.

For the CAN bus there is a module that control the messages that are sent and received as well as the format of these messages. The legs are components that provides a set of functions to represent the possible actions that can be performed and save the current status.

V. SIMULATOR

To test the system developed we have designed a basic simulator. This is a simple program that implements an interface to request a user the required parameters for a leg response message (angle of the motors and error status of the leg). With this information the program generates a CAN bus message and sends it through a pipe to the control system. This pipe is a Linux special file type and it replaces the CAN bus device file because their behavior are the same: when the destination station reads the message, deletes it from the comunication line.

This simulator is very simple but let us simulate the answers that the legs send after the execution of an order. With this simulator we can debug the system before install it in Ptinto to avoid mechanical problems and reduce the cost of software testing.

VI. FUTURE WORK

Ptinto is still in an early stage and the autonomous control system just controls its motion. There are still missing important perception systems such as cameras or optical obstacles sensors before it can completely be deployed into the abrupt environment of the Tinto river. Therefore, when Ptinto has more elements, the autonomous control system can be expanded to manage the new functions. Some possibilities are remarked next.

A. Hardware

As indicated previously, Ptinto only has sensors for collision detection by shock, which implies a low efficiency in the path to follow because the robot needs to impact before knowing there is an obstacle. With a camera or a sensor, such as optical or ultrasounds sensors, it can avoid obstacles more efficiently. Linked to this, a positional system will be very useful for the precise generation of paths by the AI system between the actual position of the robot and the goal destination.

B. Software

The actual control system can determinate the angle and the distance between two points by mathematical operations and a reference system without any relationship with the environment. This implies that the control system cannot make precise choices. In short paths the error is depreciable, but Ptinto is designed to explore areas of hundred of square meters. We also need to control the inclination of the robot, a very important fact in abrupt environments. We would need to expand the planning domain (probably by adding a new layer) to control the balance of the robot and prevent overturning.

VII. Conclusions

This paper presents a first step to build an autonomous control system for the Ptinto robot. Ptinto has been designed to be able to move around rocky and cumbersome areas in the Tinto river, one of the few areas in the world that presents some special features that make the scientists think how the Mars planet could be. The proposed architecture is based on a 3T layers. The top layer is based on the PIPSS system and the planning domain is subdivided into layers, each one operating at a different level of granularity. To test the architecture we have built a simulator that emules the Pinto behavior when moving and rotating the legs. Initially, we have just considered flat terrains.

Acknowledgements

This work is funded by the Castilla-La Mancha project PEII09-0266-6640.

References

[1] PC/104 Embedded Consortium [Online] http://www.pc104.org/

[2] J. Plaza, M.D. R-Moreno, B. Castaño, M. Carbajo and A. Moreno. *PIPSS: Parallel Integrated Planning and Scheduling System*. The 27th Annual Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG08), Edinburgh (UK), December 2008.

[3] M.D. R-Moreno, A. Oddi , D. Borrajo, and A. Cesta. *IPSS: a Hybrid Approach to Planning and Scheduling Integration*. IEEE Transactions on Knowledge and Data Engineering, 18(12):1681Ű1695, 2006.

[4] M.D. R-Moreno, D. Camacho and A. Moreno. *HPP: A Heuristic Progressive Planner*. The 24th Annual Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG05), London (UK), December 2005.

[5] A. Cesta, G. Cortellessa, A. Oddi, N. Policella, and A. Susi. *A constraint-based architecture for flexible support to activity scheduling*, AI*IA 01, 2001.

[6] T. B. Smith, J. Barreiro, D. Chavez-Clemente, D. E. Smith and V. SunSpiral. *ATHLETE's Feet: Multi-Resolution Planning for a Hexapod Robot*. In the proceedings of the workshop on Scheduling and Planning Applications of the International Conference on Automated Planning and Scheduling (ICAPS), Sydney, Australia, Sept. 2008.

[7] J. Bresina, A. Jönsson, P. Morris and K. Rajan. *Activity Planning for the Mars Exploration Rovers*. proceedings of the workshop on Scheduling and Planning Applications of the International Conference on Automated Planning and Scheduling (ICAPS), Monterey (CA), USA, 2005.

[8] J. Bresina and P. Morris. *Mission Operations Planning: Beyond MAPGEN*. In the Second IEEE International Conference On Space Mission Challenges for Information Technology, Pasadena (CA), USA, 2006. IEEE.

[9] J. Hoffmann and B. Nebel. *The FF Planning System: Fast Plan Generation Through Heuristic Search*. In Journal of Artificial Intelligence Research. 14, pp: 253-302, 2001.

[10] A. Cesta, A. Oddi, and D. Smith. *An Iterative Sampling Procedure for Resource Constrained Project Scheduling with Time Windows*. In Proceedings of the IJCAI-99, Stockholm, Sweeden, 2009.

[11] A. Gerevini and D. Long. *Plan Constraints and Preferences in PDDL3*. The Language of the Fifth International Planning Competition, Italy, 2005