Control Strategies for a Multi-Legged Hopping Robot

Rolf Allan Lüders

Submitted in partial fulfillment of the requirements for the degree of
Master of Science in Robotics.

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania

May 1, 2007

Abstract

In this dissertation, we develop locomotion control strategies for a novel multi-legged spherical robot. The conceptual robot, named *Robotic All-Terrain Surveyor* (RATS), has a spherical body roughly the size of a soccer ball, with 12 legs equally distributed over its surface. The legs are pneumatic linear actuators, which are oriented such that their axes of motion are normal to the surface of the sphere. The objective of this robot is to be capable to circumvent complex obstacles and traps through rough terrain. The 12-legged spherical version of the robot is still on its design phase and no prototype has been built yet. Instead, a simpler planar prototype of the robot is available to study the control problem. The planar robot consists of a wheel with 5 legs pointing radially, that rotates freely over the main axis of a radius arm that constraints its motion into a plane. The air-cylinders run on compressed air which is supplied over a tether that goes through the radius arm.

This work presents novel control strategies for the RATS planar robotic platform, which consist in the development of running and jumping gaits. The controls solutions presented were developed using two approaches: (1) conventional *"hand-coded"* controls, based on intuition and knowledge of the device; and (2) reinforcement learning techniques, where a policy is learned actively by maximizing the cumulative reward from a predefined reward function using value iteration. Simulators of the planar and spherical robots were implemented as tools to help the development process.

This dissertation also includes an overview of the control problem for the spherical robot. Results for this section are presented only using simulation.

Acknowledgements

I would like to especially thank my advisor David Wettergreen for his continuous support, including many insightful conversations during the development of the ideas in this thesis, and for helpful comments on the text.

I thank Professor Dimi Apostolopoulos for agreeing to let me to participate in the project from which this work is derived in the first place, and also for his exceptional guidance during all the process.

Many thanks to James Teza for his valuable help in preparing and documenting the experiments. Also for his great insights and great will as well.

I would like to express gratitude to many others that have contributed to my effort. My classmate Krishnan Ramnath for participating with me in a class project from where many of the machine learning solutions presented were developed. To the Mechanical Engineer Stuart Heys for designing and maintaining the mechanical aspects of the robot. The PhD student Clark Haynes for his great insights about the control strategies.

Contents

Chapter 1

Introduction

In this dissertation, we develop locomotion control strategies for a novel multi-legged spherical robot. The conceptual robot, named *Robotic All-Terrain Surveyor* (RATS), has a spherical body roughly the size of a soccer ball, with 12 legs equally distributed over its surface. The legs are pneumatic linear actuators, which are oriented such that their axes of motion are normal to the surface of the sphere. The objective of this robot is to be capable to circumvent complex obstacles and traps through rough terrain.

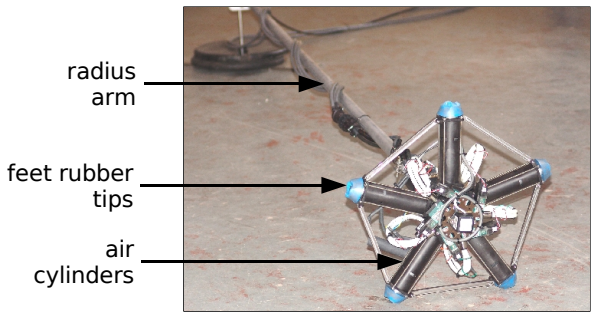


Figure 1.1: 5-legged planar robot prototype image and description.

The 12-legged spherical version of the robot is still on its design phase and no prototype has been built yet (by the time this was written). Instead, a simpler planar prototype of the robot is available to study the control problem. The planar robot consists of a wheel with 5 legs pointing radially,

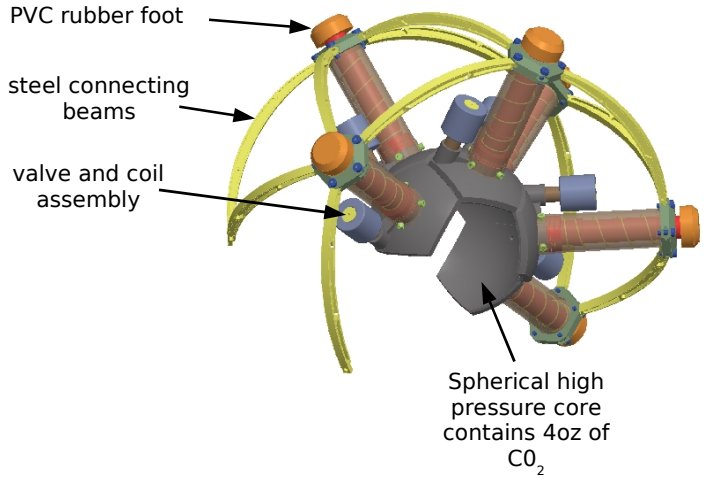


Figure 1.2: Spherical robot mechanical design: CAD drawing.

that rotates freely over the main axis of a radius arm that constraints its motion into a plane. Figure 1.1 shows the planar robot attached to the radius arm. The air-cylinders run on compressed air which is supplied over a tether that goes through the radius arm (see also Figure 3.1).

Next section describes the RATS project foundations, followed by an overview of the work.

1.1 Robotic-All-Terrain-Surveyor (RATS) Concept

The RATS project is funded by BOEING Corporation and it is developed at Carnegie Mellon University. The objective is to develop a scout and/or surveyor robot for military or search & rescue purposes. Between its capabilities, it should be able to negotiate with extreme terrains and operate for long distances.

Figure 1.2 shows a CAD breakout of the 3D robot design. The robot operates on compressed CO_2, which is stored on a central high-pressure chamber located at the center of the sphere. The spherical shape has the natural advantage that can allow the robot to roll downhill very efficiently. Also, it is inherently stable since it does not have an upright posture, and

with its 12 legs it is very likely that at least one of them is in contact with a traction surface. Its hopping ability allows it to jump or hop over obstacles that other wheeled vehicles cannot.

1.2 Controls

This work presents novel control strategies for the RATS robotic platform. Two types of behaviors are analyzed for the planar robot: running, and jumping gaits. The controls strategies are developed using two main approaches : (1) conventional *"hand-coded"* controls, based on intuition and knowledge of the device, and (2) reinforcement learning techniques, where a policy is learned actively by maximizing the cumulative rewards from a predefined reward function using value iteration.

Simulators of the planar and spherical robots are implemented as tools to help the development process. Besides the control strategies devised for the planar robot, a running strategy for the spherical robot is introduced which is based on the planar deterministic running strategy.

1.3 Thesis Outline

This work is organized as follows:

Chapter 2 discusses the related work, relating to continuous and non-continuous motion single-legged hoppers. Rolling spherical robots are also discussed.

Chapter 3 describes the planar robot design including the mechanical aspects, control hardware and low-level control software.

Chapter 4 discusses the implementation of the planar robot simulation using the Open-Dynamics-Engine (ODE) library, and the addition of a force model using a pressure dynamics model fitted with real data.

Chapter 5 presents the deterministic controllers found that allow the robot to run and jump, including results.

Chapter 6 presents the reinforcement learning controllers that perform jumping and running, including results.

Chapter 7 previews the 3D robot simulator and a preliminary running controller.

Chapter 8 unifies the different topics by comparisons and discussion.

Chapter 2

Related Work

The idea of creating a spherical robot that has extendable legs around its body seems to be very unique to RATS. However, there are systems in existence that have some individual similarities with RATS. These include single-legged hoppers and spherical rolling robots. In this chapter a brief review of these systems and their relation to RATS is presented.

2.1 Single-legged Robots

There are two main categories of single-legged robots: (1) continuous motion type, which are dynamically stabilized robots that can hop without stopping, and (2) discontinuous motion robots, that need a pause between jumps for reorientation and/or recharge.

2.1.1 Continuous Motion Hoppers

Continuous motion robots use active control systems to maintain their dynamic equilibrium and can hop in a periodic manner. The first successful planar single-legged hopper to use active stabilization was developed by K. Matsouka [19]. His robot used a high-force electric solenoid to provide thrust and was able to run over an inclined table in low effective gravity.

Matsuoka's work was followed by M. Raibert [20] at the famous Leg Lab (originally located at CMU, now it is at the MIT). Raibert's machines

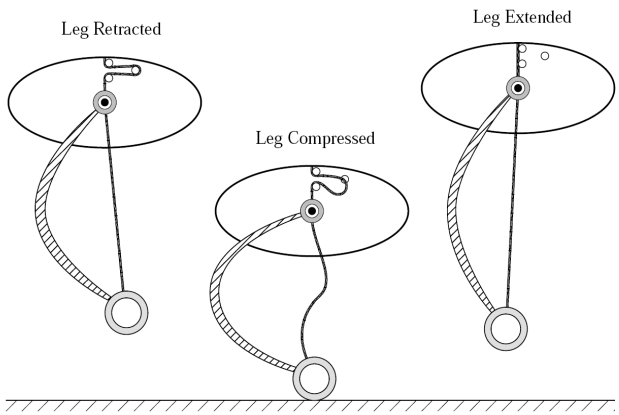


Figure 2.1: The Bow Leg Hopping Robot: Brown and Zeglin. [25].

are controlled by decomposing the control problem into three linear independent controllers: forward velocity is controlled by adjusting the foot placement; hopping height by adjusting the thrust; and pitch by changing the hip torque. This simple control scheme allowed the robot to hop in place and also run smoothly.

The first robot built by Raibert was a pneumatically actuated single-legged hopper (1980), and it used an electric motor to control the pitch of the body when the foot was in contact with the ground. The same motor was used to change the angle of the leg while in flight phase. Later he built more complex hoppers using similar control techniques including a 3D single-leg hopper, a planar biped, and a 3D biped among others.

Another milestone in single-legged dynamic locomotion is the bow leg hopping robot developed by Garth Zeglin and Ben Brown at CMU [4] [3]. Instead of using a linear actuator like a pneumatic piston, it uses a flexible bow type leg with actuators that can change the angle of the leg and also bend leg to increase the amount of energy stored while in flight phase (See Figure 2.1. The setup is energy efficient and reduces the peak power consumption since it uses the long flight phase to flex the leg again. With respect the controls, it uses planning with a physical model to find feasible trajectories.

Figure 2.2: Sandia Labs combustion powered hopping robot [24].

2.1.2 Discontinuous Motion Hoppers

Relating to discontinuous motion hopping robots, a JPL/Caltech group (*Fiorini* and *Burdick* [7], [5]) has developed several systems that use discrete leaps for locomotion. Their robots use a single-foot mechanism driven by releasing a compressed spring connected to the foot. After each jump, the spring is retracted with an electric motor. Their first generation robot had a spherical body, with the center of mass located very low near the foot location. This configuration allowed it to passively self-right after landing by rolling till the leg was in contact with the ground. A motor-controlled internal off-axis weight permitted the robot to tilt to the desired heading angle before jumping again. Later generation robots where improved by adding wheels to correct the heading before jumping and adjustable take-off angle mechanisms.

Another robot that uses a similar hopping mechanism was built at Sandia

National Laboratories [24] sponsored by DARPA. Instead of using a spring as its thrust mechanism, it uses a piston driven by a combustion chamber. They claim one of their prototypes, the size of a coffee can, is able to jump 100 times to heights of up to 20 feet on a single tank of fuel equivalent to 20 grams of combustible. Unfortunately there is not much information available about this robot and its internal combustion system.

2.2 Spherical Rolling Robots

Spherical rolling robots do not use legs or pistons for locomotion, they usually operate by changing the position of their center of mass to roll into a certain direction. Bicchi *et al.* [6] proposed a spherical robot that had a small car laying at the bottom of the sphere. Alternatively, Bhattacharya and Agrawal *et al.* [2] developed a spherical robot that had rotors inside to induce the motion of the sphere. These spherical robots are interesting nonholonomic systems whose similarities with RATS will be discussed next.

2.3 Discussion

The RATS configuration shares many particular characteristics with the systems described in this related work chapter. E.g. it uses 12 pneumatic pistons for locomotion where each piston behaves similar to the ones on the single-legged machines. When all the pistons are inside, the robot can roll freely as the spherical robots described above.

Whether RATS belongs to the class of continuous or discontinuous motion systems, the answer is that it depends on the control mode used. For obstacle jumping it can leap from rest using a single foot and jump an obstacle, corresponding to the discontinuous motion category. If the robot is running in a predetermined direction by rolling and firing certain pistons sequentially, it can be assumed the robot is operating in a continuous motion regime.

Chapter 3

Planar Robot Design

The 5-legged planar prototype of RATS is a novel design derived from the 3D 12-legged RATS concept. It is a simpler version of the 3D robot, which has proved to be extremely useful to devise and test control strategies that can be extrapolated to the more complex 3D system with minor modifications. The planar robot has also been certainly useful for the development and validation of mechanical parts such as the cylinder-piston mechanism.

In principle the planar robot is a disc containing five pneumatic actuated legs oriented radially outwards, the disc rotates over an unactuated axle and is constrained into a vertical plane of motion by a radius arm (See Figure 1.1). The wheel is connected to the articulated radius arm (boom) which holds it in alignment and allows rotation in a vertical plane. Figure 3.1 shows a CAD design diagram that describes the general setup. The actuation is given by five air cylinders powered by compressed air at 120PSI that is available through a tether. The air flow is administered to the cylinders by 5 solenoid air valves mounted on the disc. There is a slip ring mechanism to by-pass the electrical connections and the air supply. Several sensors are used to measure the state of the robot.

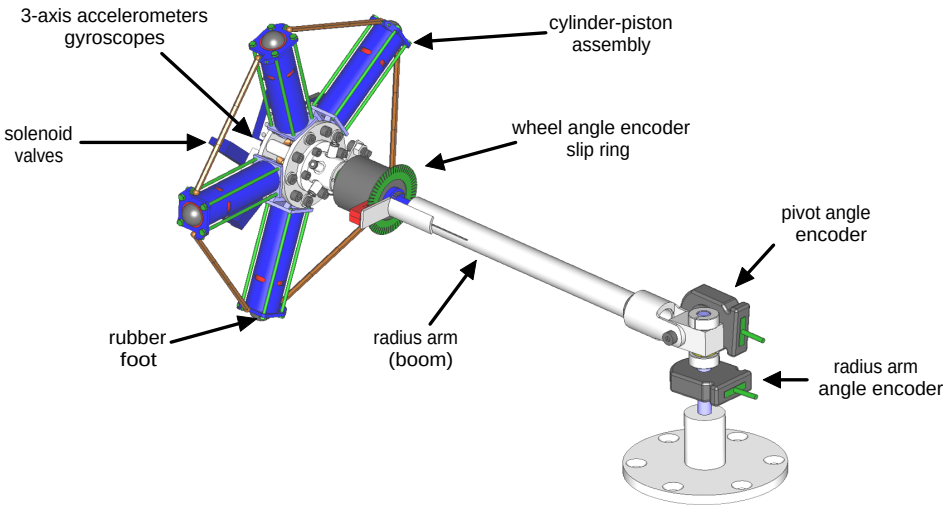


Figure 3.1: Planar robot mechanical description diagram.

3.1 Mechanical Design

Each leg consists of an aluminum base which attaches the leg to the central support structure and also holds the fitting that passes air from the valve into the cylinder. The cylinder wall, made of hard anodized aluminum tubing is squeezed between this base and an aluminum part which caps off the cylinder. This cap has a brass insert which serves as a bushing for the piston as it moves. The whole assembly is held together by tightening down four threaded rods. The honed interior of the cylinder wall provides a very smooth surface for the piston end to slide against. A spring is inserted between the piston tube and the cylinder wall to provide the return force to retract the leg. A hard stop is cushioned by an O-ring, when the piston reaches full extension. A PVC foot attached to the end of the piston provides grip and cushioning. Exhaust ports are timed to relieve pressure before the top surface of the piston makes contact, to minimize the force. A stack-up diagram of the leg components is shown in Figure 3.2.

The current leg design is a highly efficient one, the result of many piston design iterations. A honed, hard anodized aluminum cylinder wall combined

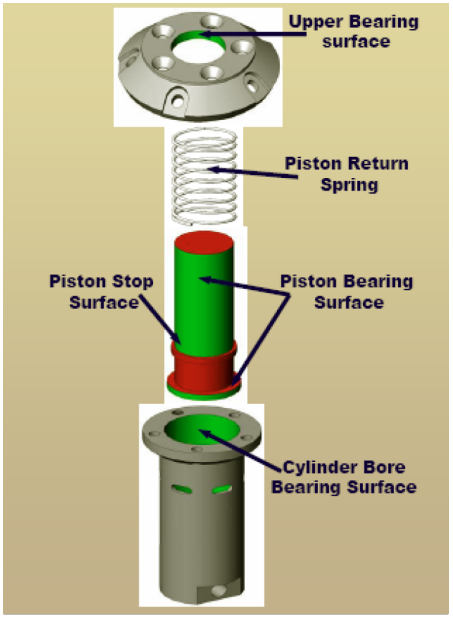


Figure 3.2: Air cylinder mechanical detail: The leg is assembled from the outside as a stack-up. The primary axial compressive force is reacted through the base. Exhaust ports are timed to relieve pressure before the piston's top surface makes contact, to minimize the force. A return spring and sliding seals allow the piston to return to its retracted position.

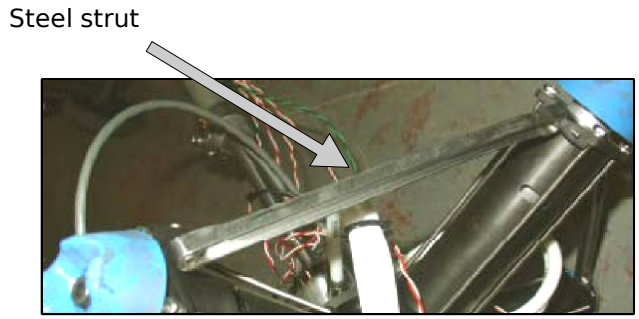


Figure 3.3: A close-up of one of the five steel struts.

with a tight fitting aluminum piston produces a gap on the order of .0001" between the two surfaces. This minimizes the amount of propellant that can escape the high pressure side of the piston while preventing the leg from binding up. This design has proven to be lightweight, highly durable (1000's of cycles under actual conditions) and surprisingly resistant to dirt.

The assembly of all five legs is held together by a cylindrical core and five steel struts, shown in Figure 1.1, which keep the legs positioned and transfer impact forces from one to another to lessen the impact on a single component. The assembly is mounted to a long axle via ball bearings and a pneumatic slip ring which allows air to continuously feed into the valves while the assembly rotates. The five valves are 3-way, low power spool valves designed for compressed air. When energized, air is passed through a short length of tubing into the cylinder. When de-energized, the remaining air in the piston is compressed by the leg return spring and allowed to pass back through the valve into the atmosphere.

3.2 Control Hardware

Several sensors are attached to the robot, these include three encoders to measure the radius, pivot, and wheel angles; two MEMS gyroscopes to measure the wheel angular velocity; and two 3-axis accelerometers to measure the accelerations of the wheel (see Figure 3.1). In order to measure a broader

Sensor	*Measurement*	*Brand*	*Model*	*Range*
gyroscope MEMS	wheel angular velocity	Silicon Sensing	RRS01-07	$\pm 1500°/s$
gyroscope MEMS	wheel angular velocity	Silicon Sensing	CRS03	$\pm 573°/s$
accelerometer 3–axis MEMS	wheel accelerations	Crossbow	CXL02LF03	$\pm 2g$
accelerometer 3–axis MEMS	wheel accelerations	Crossbow	CXL50LP03	$\pm 50g$
Encoder	wheel angle	US-Digital	E5S	200 c/rev
Encoder	radius angle	US-Digital	E5S	400 c/rev
Encoder	pivot angle	US-Digital	E5S	400 c/rev

Table 3.1: Planar robot sensors table.

dynamical range, two accelerometers and two gyros with different range capabilities are used to measure the same respective variable. The sensors and their characteristics are listed in Table 3.1.

A 3.4 GHz Pentium IV Dell desktop PC computer with 2GB of RAM runs the main control loop in software. The input signals are acquired using a multifunction input/output (MIO) board which is interfaced to the main computer via the PCI bus. The same MIO device is used to generate the actuation signals. The MIO board is a National Instruments PCI-MIO-16XE-50 and has 16 single ended or 8 differential 16-bit analog inputs; 8 digital input/output lines; and several high speed counters.

The analog signals from the accelerometers and gyros are acquired directly by the analog to digital converter of the board. The encoder pulses are counted using the internal up/down counters of the MIO using quadrature to count converters made by US-Digital (model: PC6). The external 24 volts line that powers the solenoids is switched to each solenoid using opto-isolated solid-state-relays SSR's (Opto88) which are driven directly by the digital outputs of the MIO. Figure 3.4 summarizes the connections between the sensors, actuators and the MIO board.

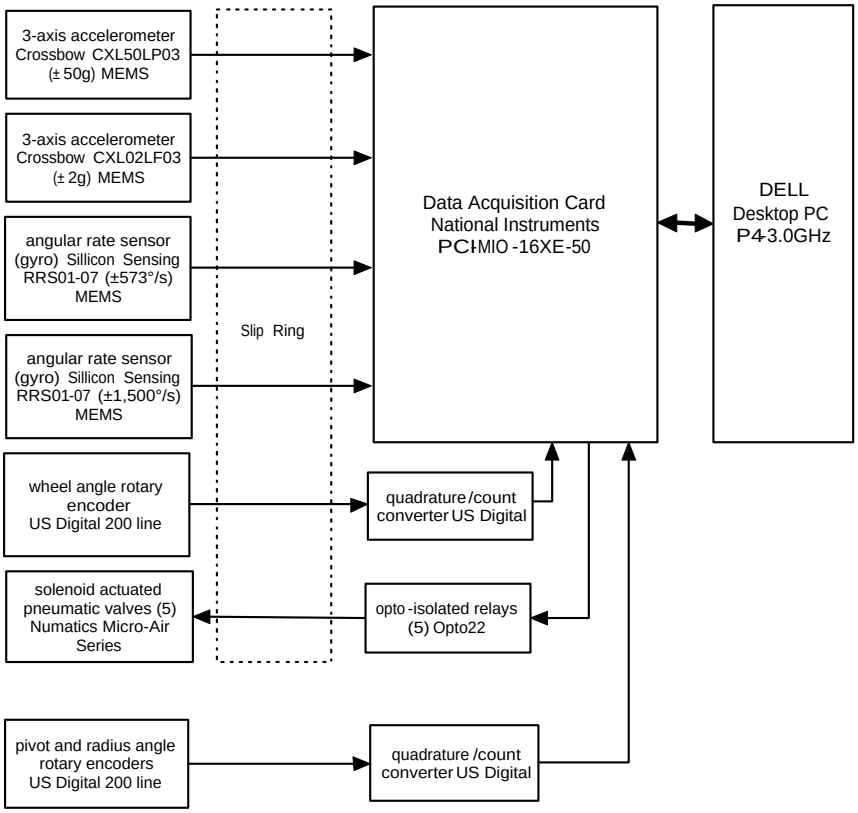


Figure 3.4: Sensors and actuators integration block diagram. The dotted box represents the slip-ring signal bypass.

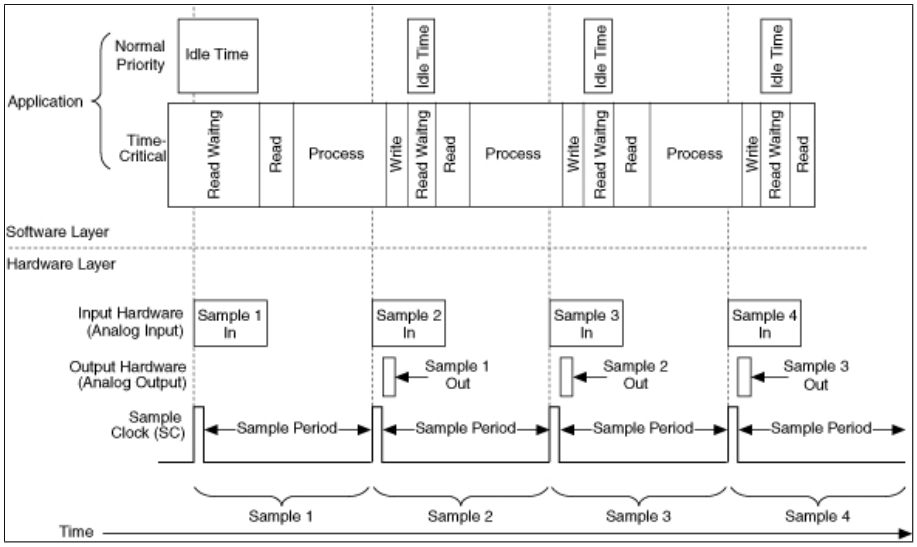


Figure 3.5: Control loop timing diagram. Hardware timed input, software timed output. *Source:* National Instruments [9].

3.3 Control Software

The host computer ran Scientific Linux 3. The control software was implemented in ANSI C language using the National Instruments *NI-DAQmx 8.0* Linux drivers. NI-DAQmx is a multi-threaded extensive controls and acquisition driver/library that takes advantage of all the functionality of the MIO board. The MIO board and drivers support single-point-real-time applications.

In our application the analog input was hardware timed and the output software timed. Figure 3.5 [9] shows the timing scheme used, where every time a hardware sample clock occurs the analog acquisition process is started. During this phase the signals from the gyroscopes and accelerometers are converted into digital signals. During the acquisition process the software loop is in wait mode. Once the acquisition is finished the software loop resumes, reading the values from the counters (encoders values) and processing all the read data to generate a new output solution. Also, every relevant variable is saved into a telemetry log on the hard drive. After the

computations are finished and the logs saved, the output is written to the digital output of the MIO board.

Although the operating system used does not support hard real time operation, the performance of the system exceeded our expectations. The control loop operated at 1KHz, logging all the relevant parameters into disk every loop. The setup used, which acquires the samples deterministically, was more than adequate for our purposes.

The Linux distribution was configured just to the bare minimum, no graphical interface or processor consuming daemons where loaded. The machine was accessed remotely using ssh, and the control loop process was given a high "soft-realtime" priority.

3.4 Discussion and Future Work

The final planar prototype proved to be mechanically very robust, this given that it must withstand severe shocks while operating. The robot showed no signs of material fatigue after countless hours of operation.

With respect the mechanical design of the actual prototype, there are some aspects that could be improved. For instance, the radius arm boom flexes considerably causing the pivot encoder to detect oscillations that do not represent the motion of the robot. This could be solved by using a stiffer tube, ideally made of composite materials to be lightweight. Another element to improve are the springs, which are not strong enough to pull back the piston effectively when the wheel is rotating and there are centripetal forces acting on the pistons.

Chapter 4

Planar Robot Simulation

A complete computer simulator of the planar robot was implemented to test new control strategies. This proved to be particularly useful to speed up the process of testing different control strategies and to understand better the dynamics of the system. The simulator was programed in C/C++ using a free open-source rigid body dynamics SDK package. The output of the simulation was viewed using an OpenGL based library (See Figure 4.1). Several phenomena was simulated including collisions with friction; joint friction; and pressure dynamics in the chambers of the air cylinders.

4.1 Physics SDK and visualization tools

The physics engine SDK used was *Open Dynamics Engine (ODE)* [22]. ODE is an excellent solution for simulating articulated rigid body structures in an interactive real-time mode. The package includes advanced features such as several built-in joint types and integrated collision detection with friction.

For visualization purposes the OpenGL-based *"drawstuff"* library included with the ODE package was used. The library had the capability to draw basic rigid bodies in a three-dimensional environment using the hardware accelerated OpenGL drivers.

The SDK package uses a first order fixed-step integrator which is fast and stable but might not be accurate enough for quantitative engineering.

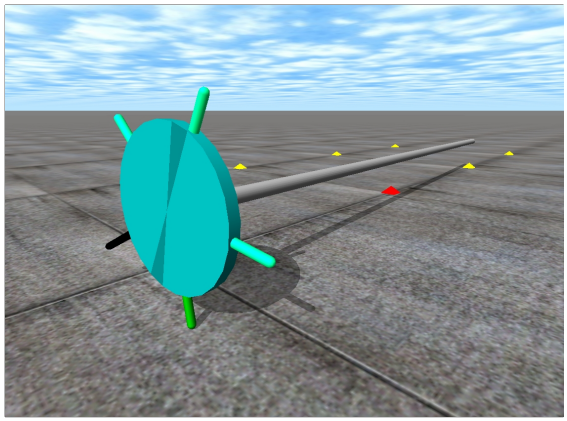


Figure 4.1: Renderized view of the 5-legged simulator.

However, the fact the simulator was fast was welcomed since it allowed us to simulate and visualize the robot in realtime. By pressing keys the legs were fired and the behavior of the robot could be observed in the screen. This was useful to devise new control strategies directly in intuitive way. In those cases where precision was important the timestep was further reduced to improve accuracy.

4.2 Simulator Implementation

The process of building the simulator in ODE consists in first creating the different rigid bodies, this includes setting their masses and moments of inertias. Then the bodies are rotated and translated to their initial positions. Finally the bodies are bonded together with a matching joint. Each body has attached a separate frame of coordinates represented by a position vector and a rotation matrix.

4.2.1 Rigid Bodies

4.2.2 Joints

It was assumed the rigid bodies were solids with particular shapes and constant densities. The wheel was treated as a sphere, and the boom and each

Body	*Weight*	*Weight Distribution*	*Dimensions*
wheel	$2.5kg$	solid sphere	radius = $0.24m$
rod and foot *each*	$0.05kg$	solid cylinder	length = $0.1m$ radius = $0.007m$
boom	$0.5kg$	solid cylinder	length = $2.12m$ radius = $0.015m$

Table 4.1: Dimensions and weights summary of the rigid bodies used for simulation.

foot-rod assembly as cylinders. The dimensions and weights were empirically measured from the real robot. Table 4.1 summarizes the characteristics of the different bodies used in the simulator.

A universal joint was used to model the constraints between the base and the boom. This special joint allows angular motion in two axes which correspond to the radius and pivot angles described before. The wheel was connected to the boom by means of a hinge joint that allows rotation on a single axis. For connecting the rod-pistons to the wheel assembly slider joints were used. Figure 4.2 describes the different joints and their locations.

4.2.3 Collision Detection

The contacts between the ground surface and the feet were simulated using the built-in collision detection with friction capabilities of ODE. The geometrical shape of the rods-feet chosen was of a capped cylinder, therefore the shape of the foot is modelled as a hemisphere. ODE includes many parameters that govern the behavior of the contacts. Between them are the coulomb friction and the softness of the surface among others. This parameters were hand-tuned to try to match the behavior and slipping of the real robot.

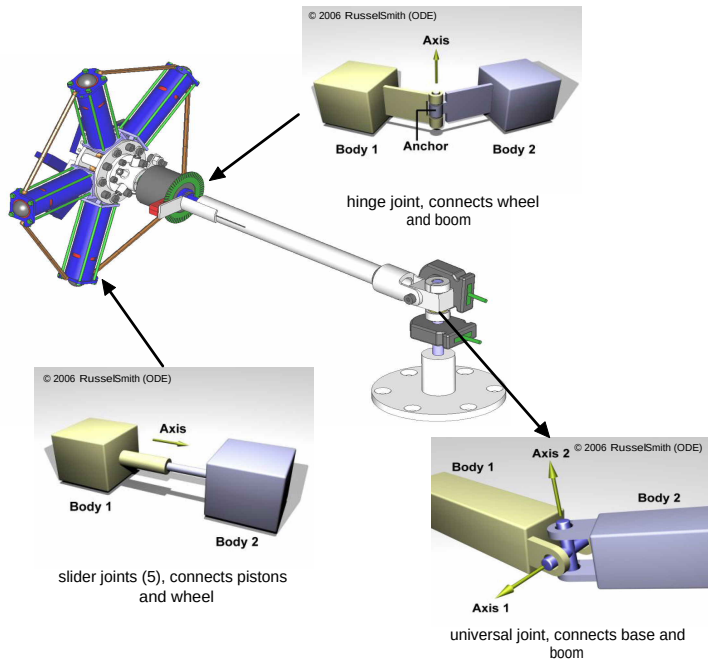


Figure 4.2: Planar robot simulator joints configuration.

4.3 Piston Force Model

The first version of the simulator assumed that the force applied by the leg was constant. Unfortunately, that assumption was largely incorrect since the chamber pressure varies significantly beacause the air is very compressible and the airflow is highly non-linear through the pneumatic components. To take this in effect, a non-linear pressure dynamics model of the complete pneumatic system was fitted. The model considered the air compressibility through the chambers; the non-linear flow through the valve; and the piston friction with the wall of the cylinder. The peneumatic model was later integrated with the main ODE simulator.

To fit the model, many of the parameters were measured directly such as weights and dimensions. Unfortunately, others like the friction and valve discharge coefficient could not be measured empirically. To find these hidden parameters a special benchtop single-legged system was used. The system replicated one of the 5-legged air cylinders and had extra sensors to acquire information such as the pressure inside the chamber and position of the piston.

4.3.1 Non-Linear Model

The non-linear model is based on the work from Richie and Hurmuzlu in [21]. The model takes in consideration the air compressibility; non-linear air flow through the valve; and the air line delay and attenuation among others. The model can be separated into four sections: Piston-load dynamics; cylinder chamber thermodynamics; connecting tube; and the orifice valve model. The four sections mentioned are discussed next.

Piston-Load Dynamics

The model proposed by Richie and Hurmuzlu was designed for *double-acting* air cylinders, this means it considers two chambers, one for extending and the other for retracting the rod. However, the RATS air-cylinders are *single-acting*, so they use one pressurized chamber for extending and a spring for

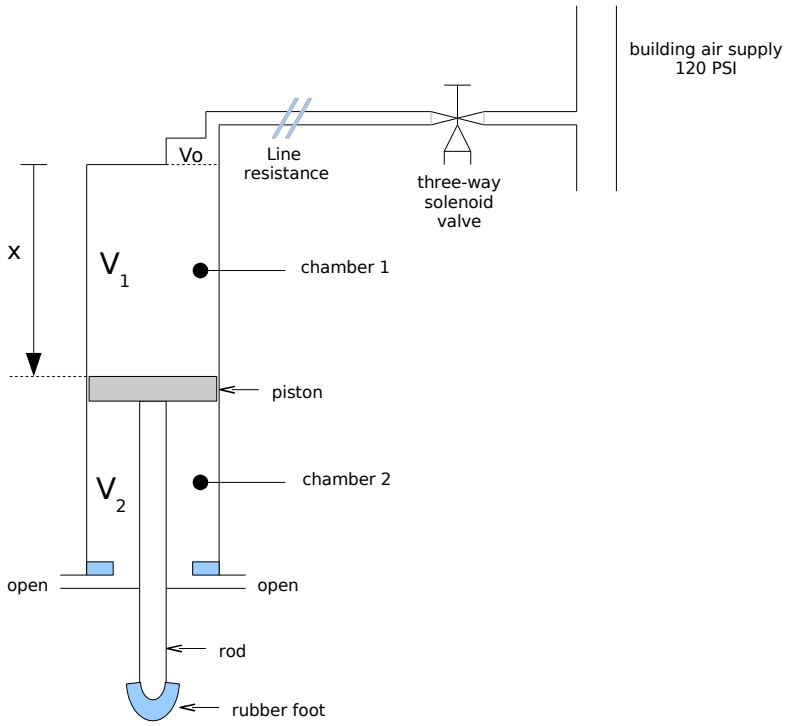


Figure 4.3: RATS single air cylinder pneumatics diagram.

retracting. To adapt the model to the RATS configuration, one of the chambers of the model was assumed to be open to the atmosphere. Figure 4.3 shows a schematic diagram of the RATS air cylinder, the lower chamber (chamber 2 in the figure) is open through two massive exhaust ports, and the upper chamber is connected to the three-way valve.

The equation of motion can be expressed as the following,

$$(M_L + M_P)\ddot{x} + \beta\dot{x} + F_f + F_L = P_1A_1 - P_2A_2 - P_aA_r$$

Where M_L is the mass load, M_P the rod-foot-piston mass, F_f the Coulomb friction, F_L the load force β the viscous friction coefficient, P_1 and A_1 are the pressure and area of chamber one, P_2 and A_2 are the pressures of chamber two. P_a is the ambient pressure and A_r is the rod area.

Cylinder Chamber Thermodynamics

The volume inside the cylinder can be represented by,

$$V = V_o + Ax$$

and,

$$\dot{V} = A\dot{x}$$

From the volume formula above, A is the sectional *area of the piston*, and x is the *position of the piston* as shown in the figure. Since the process is considered adiabatic, the variation of the pressure inside the chamber is given by,

$$\dot{P} = \frac{\kappa RT}{V}(\dot{m_{in}} - \dot{m_{out}}) - \frac{\kappa P}{V}\dot{V}$$

from where κ is the *specific heat ratio*, T is the *temperature* and $\dot{m_{in}}$ and $\dot{m_{out}}$ are the input and output *mass flows*.

Connecting Tube

The connecting tube couples the valve with the pneumatic cylinder. The mass flow through the connecting tube is modelled by:

$$\dot{m}_t(L_t,t) = \left\{ \begin{array}{ccc} 0 & if & t \leq \frac{L_t}{c} \\ \exp\left(\frac{R_t RT}{2P}\frac{L_t}{c}\right) & if & t > \frac{L_t}{c} \end{array} \right\}$$

Where L_t is the *length of the tube*, c is the *speed of sound*, and R_t is the *tube resistance*.

Orifice Valve Model

This is probably the most important restriction in the system. The model of the valve is described by the following formula:

$$\dot{m}_v = \left\{ \begin{array}{ccc} C_f A_v C_1 \frac{P_u}{\sqrt{T}} & if & \frac{P_d}{P_u} \leq P_{cr} \\ \\ C_f A_v C_2 (\frac{P_u}{\sqrt{T}})(\frac{P_d}{P_u})^{1/\kappa}\sqrt{1-(\frac{P_d}{P_u})^{(\frac{\kappa-1}{\kappa})}} & if & \frac{P_d}{P_u} > P_{cr} \end{array} \right\}$$

Where P_d and P_u are the *upstream* and *downstream* pressures respectively, A_v is the *valve opening area*, C_f is a nondimensional discharge coefficient, C_1 and C_2 are model constants, and P_{cr} is the *critical pressure ratio* at which the discharge mode changes from linear to a non-linear regime.

Simulation

To simulate the motion of the piston the four modules detailed above were put together to form two first order differential equations. One of them gives the pressure change, and the other, the piston linear acceleration. The pressure change depends on the position and linear velocity of the piston, while the acceleration of the piston depends on the pressure. This means both equations are nested so they need to be solved at the same time.

The equations were solved numerically using a simple fixed-step integrator written in Matlab.

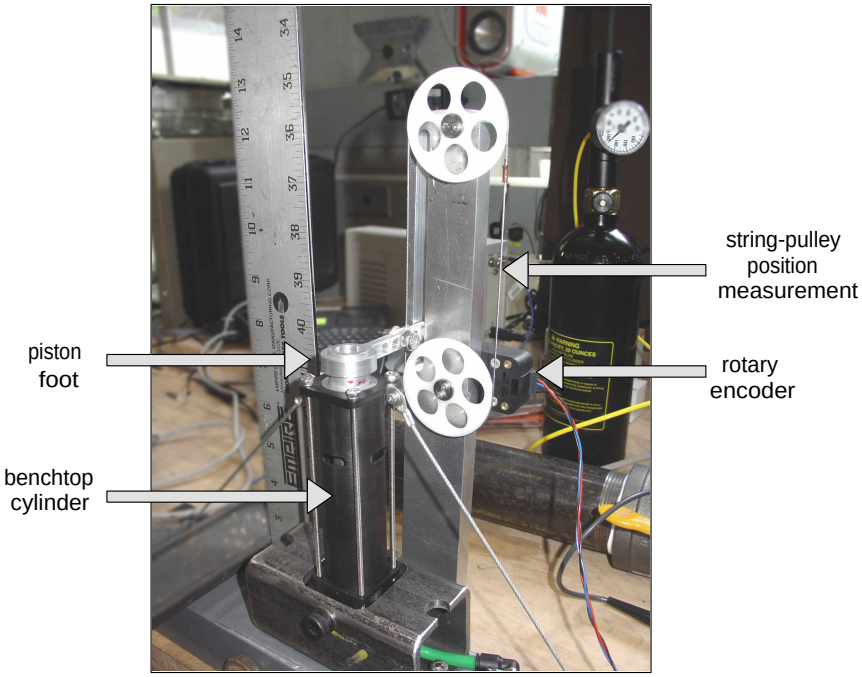


Figure 4.4: Benchtop single-piston setup.

4.3.2 Single-Leg Benchtop System

To search for the hidden parameters a single-legged benchtop setup was used, the system is shown in Figure 4.3. The characteristics of the valve and air-cylinder were the same as those of the 5-legged planar robot. The air supply consisted of a small tank of high-pressure *C02* coupled to a pressure regulator. This allowed to select any pressure between 0 and 750 *PSI.* The weight was varied by laying on the exposed rod-end different weights. There was an encoder-based mechanism that allowed the linear motion of the piston to be measured, and a piezoelectric pressure sensor inside the chamber to sense the pressure.

The piezoelectric pressure sensor and encoder counts were acquired using an oscilloscope. The valve was energized with a manual switch which also triggered the oscilloscope.

4.3.3 Model Fitting

Figure 4.5 shows the data collected with the benchtop air cylinder. The dotted line corresponds to the position of the piston, and the continuous line to the absolute chamber pressure. It is interesting to note the initial delay of around *10 ms* that takes for the chamber to commence to pressurize. That is probably due to the time it takes for the valve to actuate. Also, the profile of the pressure is highly non-linear, it builds-up very quickly and then decays as the piston adquires a higher velocity. Once the piston goes past the exhaust ports the pressure drops to atmospheric levels. This proves that the force applied by the piston is definitely not constant since the force is directly proportional to the chamber pressure.

Data from 18 single firings experiments was used to fit the model. Each experiment was done with one of four different weights we had available, that ranged from *0.477 Kg* to *5.68 Kg* and with one of the five different preset supply pressures which fluctuated from *60 PSI* to *150 PSI.* The range of weights and supply pressures were chosen to match the possible operating conditions of the 5-legged RATS.

Each experiment consisted in logging the pressure of the chamber and

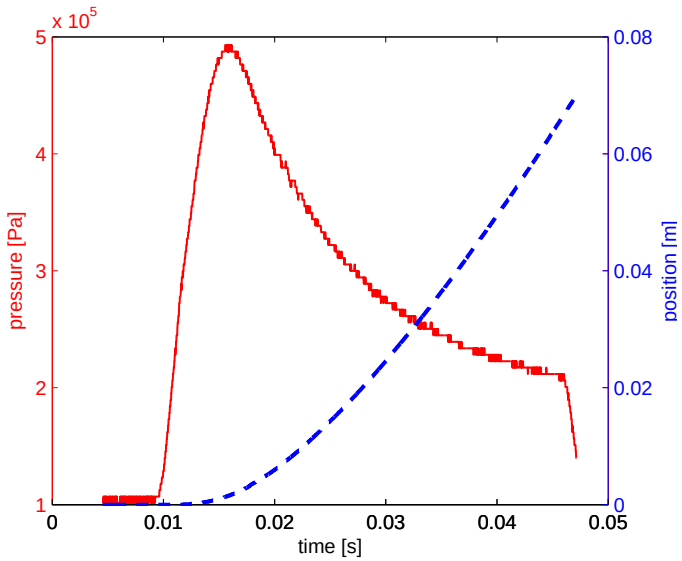


Figure 4.5: Results from a single firing experiment done with the single-legged benchtop setup. The *dotted* line corresponds to the position of the piston, and the *solid* line to the absolute chamber pressure. The load weight was of *2.245 kg* and the supply pressure was regulated at *80 PSI*.

Parameter	*Value*	*Units*
viscous friction β	5.2843	$[kg/s]$
Coulomb friction F_f	9.7003	$[N]$
discharge coefficient C_f	0.5378	–
valve orifice radius R_v	1.0215	[mm]
line length L_t	0.0	$[m]$
end of stroke inactive volume V_0	$5.4 \cdot 10^{-6}$	m^3

Table 4.2: Model parameters found using the optimizer.

the linear position of the piston starting from when the valve was energized. The sequence recorded included the motion of the piston from its retracted rest position to the the completely extended position after having hit the hard-stop.

A multi-variable unconstrained non-linear optimizer that operated with a single error metric was used for finding the hidden parameters. The error represented the differences between the measured values and the simulated values. First, the friction coefficients were found using only the piston-load dynamics model by assuming the pressure was known. Once the friction coefficients were obtained they were used as constants for finding the pressure model. This two stage process was used to reduce the degrees of freedom in the search. Choosing adequate initial values for the parameters was fundamental to avoid local minima and to converge to an acceptable solution, therefore several guesses were tried.

The optimization was done in Matlab using the built-in *fminsearch()* function. The 18 experiments were used to find a single optimal solution.

Table 4.2 lists the parameters that were found using the optimizer. The line length is zero because when set to an initial positive value, the optimizer would decrease it to values bordering zero, sometimes positive and sometimes negative depending on the initial values of the other parameters. Given this, it was fixed to zero and removed from the variable list the optimizer could change. The probable reason for this is that the actual line

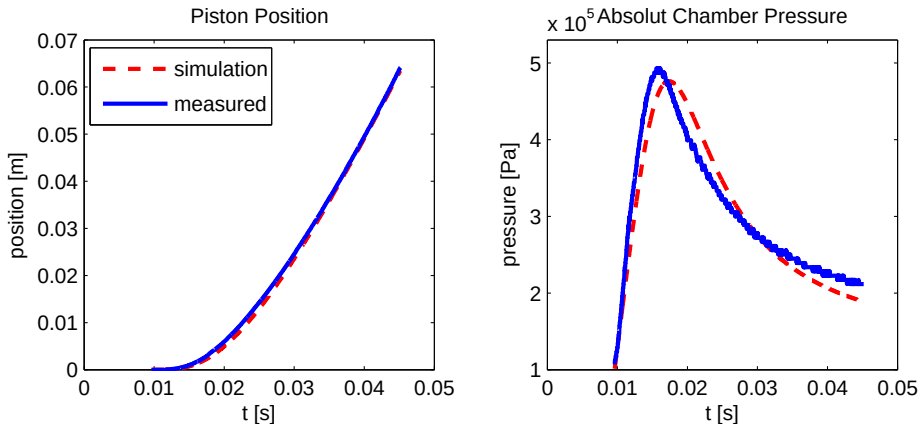


Figure 4.6: Comparison of a simulated run *(dotted line)* versus a real run *(solid line)*. The *left* plot shows the piston position, and the *right* plot the chamber pressure. The load used was of *2.245 kg* and the supply pressure was regulated at *80 PSI*.

that connects the valve and the cylinder is extremely short, of less than one inch, therefore a value of $L_t = 0$ is not a surprise. If there was some line attenuation it was probably taken by the valve restriction C_f parameter.

4.3.4 Validation

Figure 4.6 is a comparison of the simulation and the data acquired from the single-leg benchtop. The case chosen, with a pressure of 80 PSI and a load weight of 2.245 kg, was randomly picked from the 18 experiments for display purposes. It can be seen the simulator tracks real values closely. The same comparison for different weights and supply pressures gives similar results.

4.3.5 Force Model with the 5-legged Robot Simulator

Adding the model to the simulator was a straightforward process. The force applied by the piston was calculated at every time step once it was fired according to the following formula:

$$F[t] = P[t] \cdot A_1 - F_f - \beta \dot{x}[t]$$

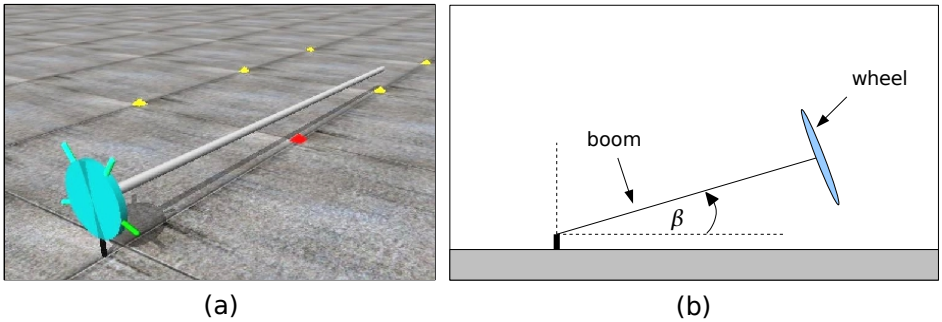


Figure 4.7: Jumping vertically on one foot experiment. (a) is a renderized view of the simulator, with the robot at rest position with foot number 1 on the floor; (b) is a digram of the test performed, and shows the robot in the air after the valve was energized. The angle β is the pivot angle.

,were $P[t]$ corresponds to the relative pressure in the chamber and it is calculated using the dynamic pressure model. The pressure model uses as inputs the piston position and velocity which are both provided by the ODE simulator.

Results: Comparison of the Behavior of the Enhanced 5-legged Simulator with the Robot

In this section, a test to demonstrate that the force model operates correctly in the 5-legged simulator is presented. Both, the simulator and robot were subject to the same test. The test consisted in jumping verticaly over one leg from rest. The pivot angle against time was compared for both systems. Figure 4.7 shows the simulator graphical view *(a)*, and also a diagram of the test *(b)*. For each trial the valve was opened for different time lengths.

Figure 4.8 shows the results of the test for 4 different valve opening times *(20, 30, 40, 40 ms)* . The simulator tracks the real behavior very accurately. Also, is relavant to notice that the maximum height of each trial depends on the valve opening time. For longer valve opening times the higher the robot gets. This occurrs since for smaller opening times the valve closes before the piston has reached its limit, therefore the power stroke is shorter.

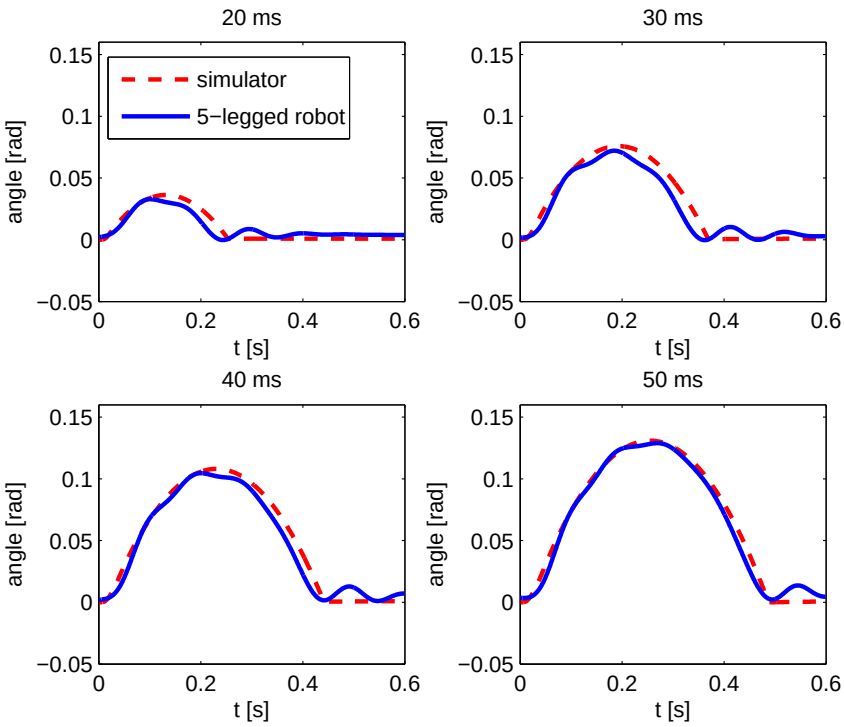


Figure 4.8: Pivot angle for the simulator *(dotted line)* and the 5-legged robot *(solid line)* for 4 different trials with different valve opening times. The pivot angle of the robot was measured with the pivot encoder.

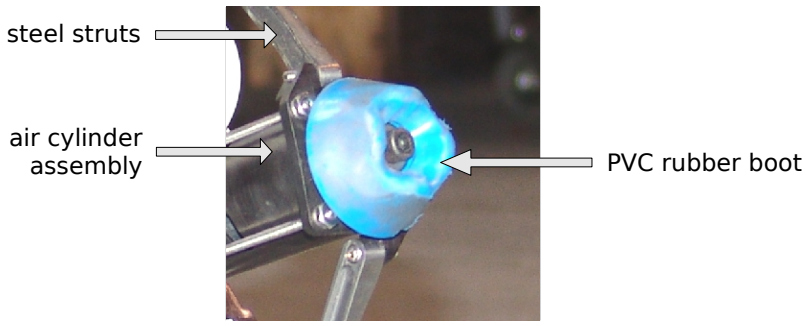


Figure 4.9: RATS PVC rubber foot close-up.

The encoder signal is not a smooth as that of the simulator result since the pivot encoder is located at the base of the radius arm, which flexes when the wheel moves vertically. In fact the signal shown is already filtered using a zero-phase lowpass filter to remove some of the higher frequency wiggling.

4.4 Discussion and Future Work

Even though the accuracy of the simulator was greatly improved with the addition of the pressure model, there is still room for improvement. A possible significant source of error is the modelling of the contacts between the surface and the feet of the robot. The simulator models the geometrical shape of the feet as hemispheres, while the real shape is shown in Figure 4.9, which is a much intricate shape in comparison. To simulate this, the geometrical shape of the real PVC rubber foot would have to be incorporated to the simulator. ODE has development tools that allow to add new geometrical shapes using triangular meshes. Another aspect of the contacts that needs to be analyzed is the friction and deformation of the foot.

Chapter 5

Planar Robot Control Gaits

In this chapter control gaits devised for the 5-legged robot are presented. Two main categories are presented: running and jumping gaits. Running gaits are meant for displacements in reasonably flat surfaces while jumping gaits are designed to hop or climb over discrete obstacles.

The control gaits presented in this chapter were found empirically using discrete controllers. Conversely, in Chapter 6 reinforcement learning techniques are used to try to find control policies with no previous knowledge of the environment or robot.

The running gaits developed allow the robot to accelerate from rest to $3.2ms$ in less than 3 seconds. Two controllers are presented, one that relies on an open loop policy to update parameters of an underlying primitive controller, and a second version that uses velocity feedback instead to update the same parameters, thus being more robust against disturbances.

The jumping controller relies on the running strategy to acquire horizontal velocity, and after that uses a firing angle control scheme to apply the vertical thrust to perform the jump using a single leg.

5.1 Running

The running gait is based on the rolling motion of the wheel. The legs are fired sequentially once each is in contact with the ground at a certain angle

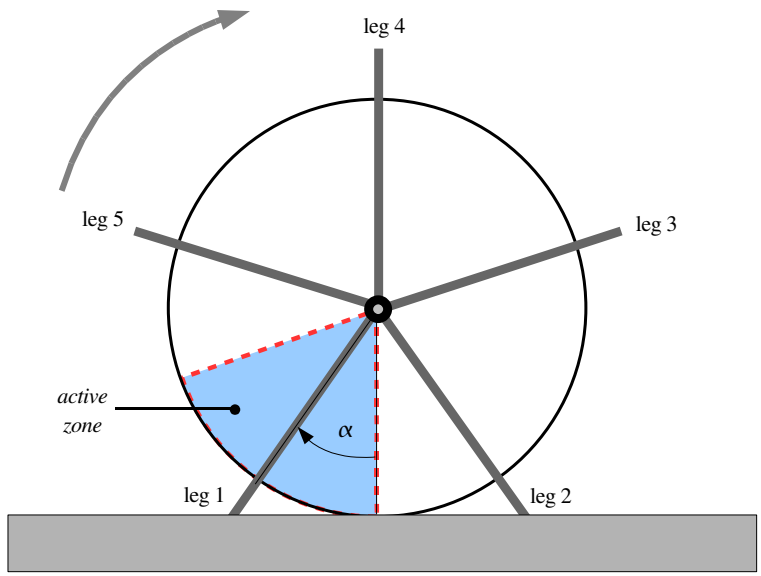


Figure 5.1: Description of the main parameters of the running gait. The shaded zone corresponds to the *active zone*, and the α angle is defined between the normal of the surface and the leg (any of the 5) that is inside the *active zone*. Each leg is fired at a predetermined α once it enters the fixed *active zone*.

from the normal of the surface. Each time a leg is fired it applies horizontal and vertical forces that induce the rotation of the wheel into a predetermined direction. Due to the rolling motion, the legs are positioned sequentially in the correct angle/position for being fired.

5.1.1 Underlying Running Controller

To understand the basic running controller, lets assume the robot is at rest as shown at Figure 5.1, on a flat surface with legs 1 and 2 in contact with the ground surface. If we want the robot to rotate clockwise (arrow direction in the figure), leg one needs to be fired first. This should force the wheel to roll clockwise and leg 2 should eventually reach to the same angle with respect the surface as leg 1 initially. Once that occurs leg 2 is fired and leg 3 is the next to be fired and so on.

To implement this intuitive controller, an *active zone* is defined as shown in Figure 5.1. The *active zone* translates with the wheel but does not rotate with it. The angle *alpha* (α) is defined as the angle between the normal of the ground surface and the leg that is inside the *active zone* as show in the figure. The circumference length that defines the *active zone* is the same as the one defined between two legs ($2\pi/5$), therefore there will be always a single leg inside the *active zone*. We define the *alpha-fire* (α_f) angle as a predefined α at which the leg is fired after the legs enters to the *active zone*. For the robot to run as intended, the α_f needs to be carefully chosen, since firing too early (small α_f's) will make the robot to hop vertically and firing too late will cause part of the stroke to be lost in the air because of the rolling motion of the wheel. This issues will be analyzed in future sections when the final running gaits are presented.

In case the robot needs to roll in the counterclockwise direction (opposite to the arrow shown in Figure 5.1) then the *active zone* is just relocated by mirroring it to the right side of the wheel.

This controller was implemented using a state machine which fired the leg for a determined period of time once the leg was in the *active zone* once the α_f had been reached. The action of firing a leg consists in opening the pneumatic valve for a predefined amount of time. We call this time *valve-opening-time*(t_v). As seen in Section 4.3.5, the longer the valve remains open, the longer the the power stroke will be. Naturally, leaving the valve open too long once the piston is located past the exhaust ports will not make any difference.

Figure 5.2 shows the results from the experiment presented in Section 4.3.5, where the t_v is varied from $20ms$ to $60ms$ and the height of the jump is analyzed. The plot shows the relation between the maximum height of the jump (y-axis) versus the t_v time (x-axis). Note that for t_v's of over $55ms$ there is no noticeable improvement because the piston already reached the exhaust ports therefore the pressure drops to the atmospheric level. These results are important since they prove that the energy injected by the legs can be modulated by varying the t_v time.

The low-level controller presented is then governed just by two parame-

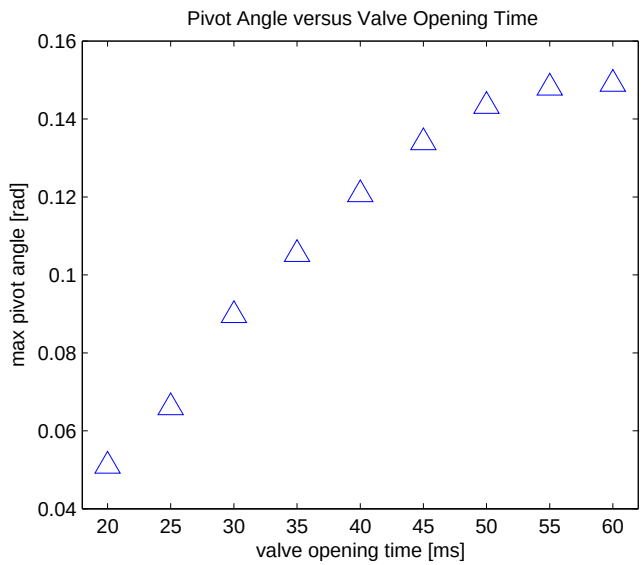


Figure 5.2: Maximum pivot angle reached when hopping in one leg with different valve opening times (t_v).

ters, the α_f angle the t_v time interval. All the running gaits presented next are build on top of this underlying controller.

5.1.2 Fixed Angle-Offset Running Gait

The fixed angle-offset controller was the first that made the robot run fast and smoothly. It was basically the low-level running controller with a constant α_f and t_v. The wheel angle was the only state variable used to control the robot, and it was used to calculate the α angle. The final $\alpha_f = 0.1$ [rad] and $t_v = 40$ [ms] values that allowed the robot to run smoothly were found by trial and error with the robot since the simulator was still under development.

The first interesting insight was that the robot could not start itself and accelerate to a smooth running mode with a fixed α_f. Therefore a short hand impulse was given to the wheel in the rotation direction for the robot to acquire enough momentum to reach the smooth and fast running mode and stay stable in that mode.

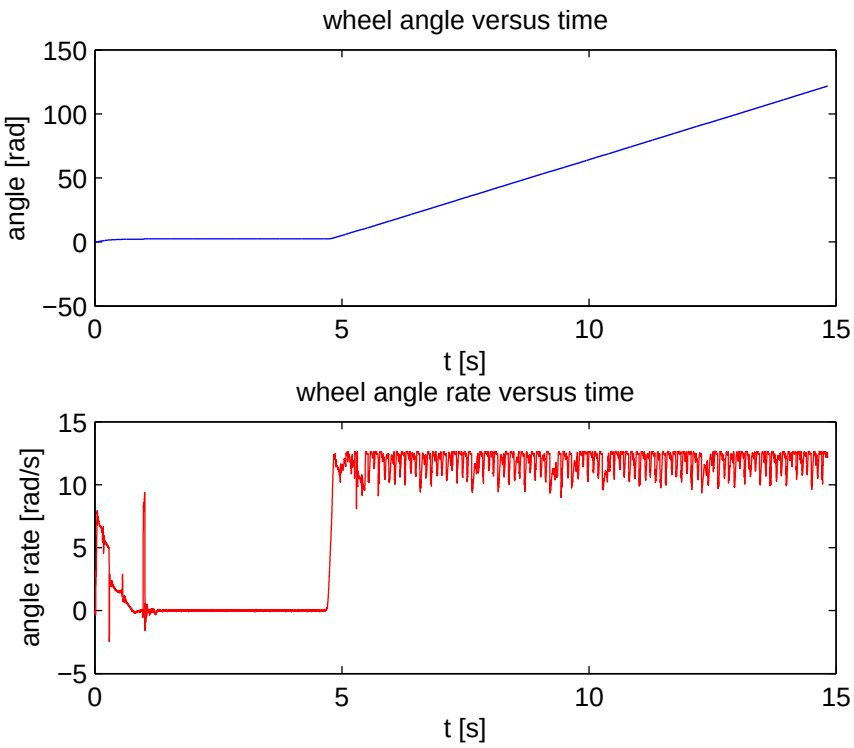


Figure 5.3: Fixed α_f and t_v control. Wheel angle versus time *(top)* and wheel angle rate versus time *(bottom)*.

Figure 5.3 presents the running results of the experiment described. The wheel angle rate shows an initial movement but it then stalls. After 5 seconds a hand impulse was given and the robot started to run smoothly as it can be seen on the plot. The wheel angle rate was acquired with the onboard gyroscope but was not used for control purposes.

An improved controller was written later that allowed the parameters α_f and t_v to be varied using the keyboard in real time while the robot was running. This feature was very useful to understand better the influence of the two parameters in the behavior of the system. By varying the parameters it was found that the top speed was reached at $\alpha_f \approx 0°$, this means that the valve starts to energize when the leg is vertical in contact with the ground surface. Another interesting discovery was that the robot could start by itself if a large α_f was used at the beginning and then the value was lowered as the wheel gained momentum. This concept will be used by the controllers presented next.

The video Fixed-Offset Control Running [11] shows RATS running while the parameters α_f and t_v are varied manually using the keyboard of the computer. The video shows that the robot can start itself, and accelerate to the smooth running gate, this was done by decreasing α_f manually. Also the t_v was varied, affecting the speed of the robot, and it can be distinguished by the change in the sound which becomes much quieter when the t_v is smaller. This occurs because for small t_v's *(below 30 ms)* the piston never gets past the exhaust ports when running.

5.1.3 Variable Angle-Offset Gait

Instead of using a fixed α_f angle, we introduce controllers that automatically vary this angle to accelerate fast from rest without intervention (before somebody had to tweak the α_f with the keyboard for the robot to start, or give it an initial hand impulse). The first proposed controller reads the value of this angle in an open-loop mode by evaluating a function with respect to time. The second controller is a closed-loop solution that derives the angle α_f from the wheel angle rate.

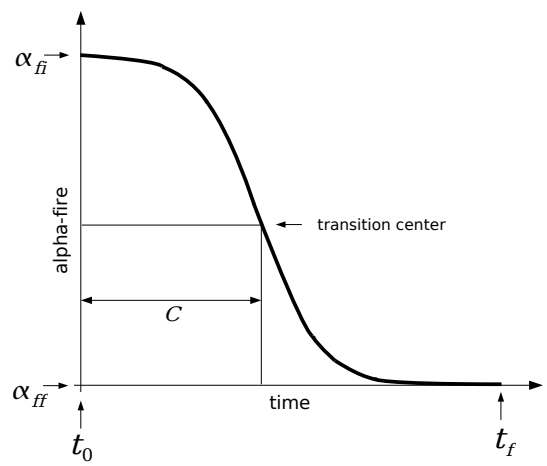


Figure 5.4: Time controlled α_f transition formula.

Open-loop α_f

An arbitrarily smooth function was used to transition from the initial alpha-fire ($\alpha_{\mathbf{fi}}$) to the final alpha-fire ($\alpha_{\mathbf{ff}}$). The constants $\alpha_{\mathbf{fi}}$ and $\alpha_{\mathbf{ff}}$ were found experimentally. $\alpha_{\mathbf{fi}}$ makes the robot to start reliably from rest, and $\alpha_{\mathbf{ff}}$ causes the robot to run fast and smoothly. The function used is given by the following formula,

$$\alpha_f(t) = A \tanh(Bt - C) + D$$

Figure 5.4 shows the shape of the function presented above. This particular function was selected because it does a smooth transition and it is highly customizable through few parameters.

The parameters A and D can be found given that $\alpha_f(t_0) = \alpha_{\mathbf{fi}}$ and $\alpha_f(t_f) = \alpha_{\mathbf{ff}}$ then,

$$A = \frac{\alpha_{\mathbf{ff}} - \alpha_{\mathbf{fi}}}{2}$$

$$D = \frac{\alpha_{\mathbf{ff}} + \alpha_{\mathbf{fi}}}{2}$$

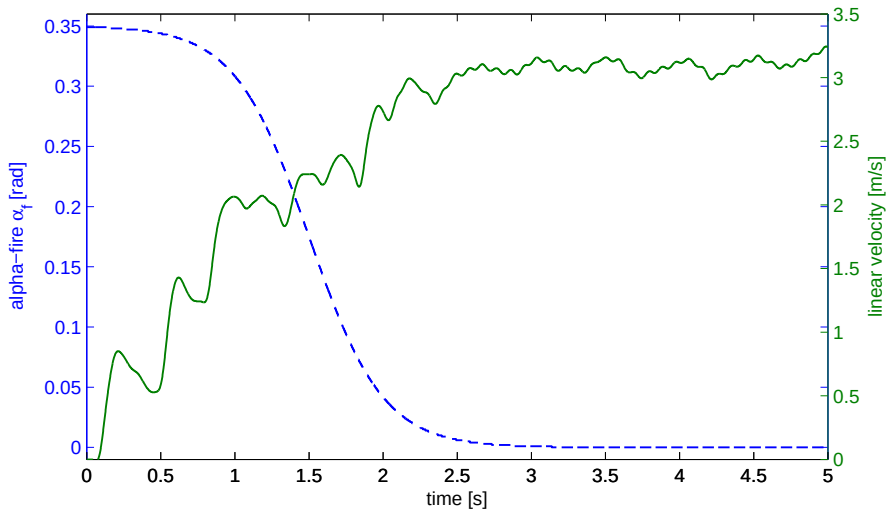


Figure 5.5: Accelerating from rest using the open-loop control results. The plot shows the α_f *(dotted line)* and the linear velocity *(solid line)* versus time.

B is the slope of the transition, and C is the lapse of time before the center of the transition occurs. By using the simulator the parameters B and C were tuned for fast acceleration. The controller was then tested in the real robot and it worked adequately. The parameters found after a brief tuning were: $B = 2.1$ (the slope), and $C = 3.0$ seconds using the real robot.

Figure 5.5 shows that the wheel accelerates fast from rest in less than 3 seconds. The robot is capable of achieving a final speed of around $3.2m/s$ which is impressive for such a device. The video Variable-Offset-Angle Open Loop Control Running [18] shows how this controller works, by starting and stopping several times to show the acceleration.

Closed-loop α_f

One of the main drawbacks of varying the α_f in an open-loop mode is that is not robust against perturbations. The controller presented next uses the fact that for accelerating, the correct α_f is highly related to the angular rate of the wheel (ω). Intuitively this is very similar to what happens in an internal

combustion piston engine, were the firing is advanced proportionally to the speed. This is done to shift the maximum expansion power into the optimal time, which is basically when the piston starts to return. The controller proposed does something very similar in the robot by causing the maximum force of the piston to be applied always when the leg is inside the active zone at a predefined angle. This accounts for the delays of the valve to open and of the maximum pressure to build.

Initially, a single linear relation between ω and α_f was used, the parameters of the function were found by using the same $\alpha_{\mathbf{fi}}$ and $\alpha_{\mathbf{ff}}$ constants from the open loop controller, and additionally by the the max angular velocity $\omega_{\mathbf{f}}$ which was found experimentally. The linear function used was,

$$\alpha_f(\omega) = A\omega + B$$

with,

$$A = \frac{\alpha_{\mathbf{ff}} - \alpha_{\mathbf{fi}}}{\omega_{\mathbf{f}}}$$

$$B = \alpha_{\mathbf{fi}}$$

This linear relation did not work because the slope was too steep at the beginning. The robot would fire a couple of legs but then stall. To solve the problem a non continuous function was tried, it was made up of two linear functions concatenated.

$$\alpha_f(\omega) = \left\{ \begin{array}{lll} C\omega + D & if & \omega < \omega_s \\ \\ E\omega + F & if & \omega \geq \omega_s \end{array} \right\}$$

with,

$$D = \alpha_{\mathbf{fi}}$$

$$E = \frac{\alpha_{\mathbf{ff}} - \alpha_{\mathbf{fi}}}{\omega_{\mathbf{f}}}$$

$$F = \omega C + D$$

ω_s was the switching angular velocity. The slope of the initial function (E) needs to be smaller than the B of the original single linear function for this to work. This solution worked reliably, and the robot would accelerate fast to the running mode. Also it could accept perturbations as changes in the inclination of the ground surface. In the Variable-Offset-Angle Closed Loop Control Running [17] video the controller is shown and the robot was tested by manually applying a force to the boom mast to try to stop it, proving that the robot adapts without stalling, and once the force is released the robot resumes to the fast running mode.

The ω_s was found experimentally by trying different values, the same for E, the slope of the starting linear function. There are better ways of optimizing this parameters using learning techniques. Also, it is probably better to use a single parametrized non linear function to represent the relation $\alpha_{f(\omega)}$ than using two linear functions. All this improvements are proposed as future work.

The angular velocity of the wheel (ω) was acquired using the MEMS gyroscopes on board. The signals measured had a high frequency component that is generated every time a leg touched the ground surface or the leg was fired. A 4th order FIR filter was used to remove some of this noise improving the results. This high frequency oscillations do not represent the "average" angular momentum of the wheel.

Figure 5.6 shows the results of the new controller in the robot. The α_f signal is noisy because it is proportional to the wheel angular rate. Unfortunately, no further filtering of the angular velocity was done to avoid an excessive delay in the signal. Another relevant aspect from the results is that the close-loop controller actually outperformed the open-loop controller, converging to slightly negative α_f's. This means the valves were being energized slightly before the correspondent leg was normal to the ground surface. To allow this, the active zone was rotated slightly counterclockwise to allow for negative α_f angles.

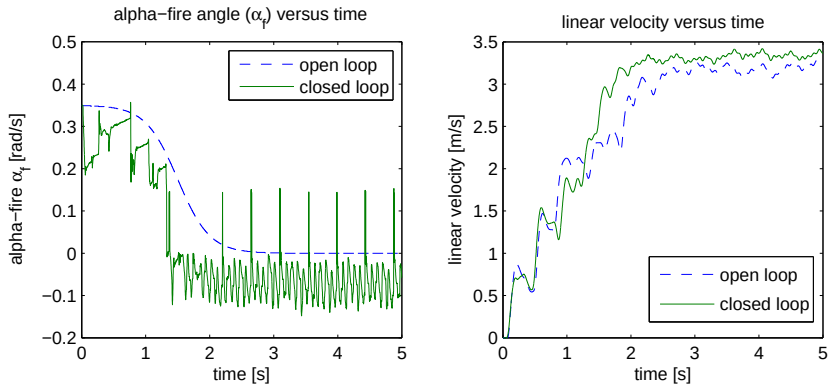


Figure 5.6: Accelerating from rest using the open-loop α_f control results. Results of the close-loop controller *(solid line)* are compared with ones of the open-loop controller *(dotted line)*

5.1.4 Initialization

Since the wheel angle sensor was a relative angle encoder, the MEMS accelerometer was used to measure the gravity vector and thus determine automatically which two feet were in contact with the ground surface every time the algorithm was initialized.

5.2 Jumping

In this section the jumping strategy is presented, which consists in controlling the robot to allow it to jump over obstacles, or steps. The jumping sequence consists in first acquiring horizontal speed by running using the running strategy. Once the robot is near the obstacle, it lands on one leg which acts as a hinge joint with the ground surface, and at a predefined angle the leg is fired to achieve a parabolic trajectory in the air to jump over the obstacle. Figure 5.7 shows the sequence of events that occur before jumping.

First, a dynamics model of the robot with a leg on the ground surface is presented. Following that, the control scheme is implemented in the robot and the results are presented.

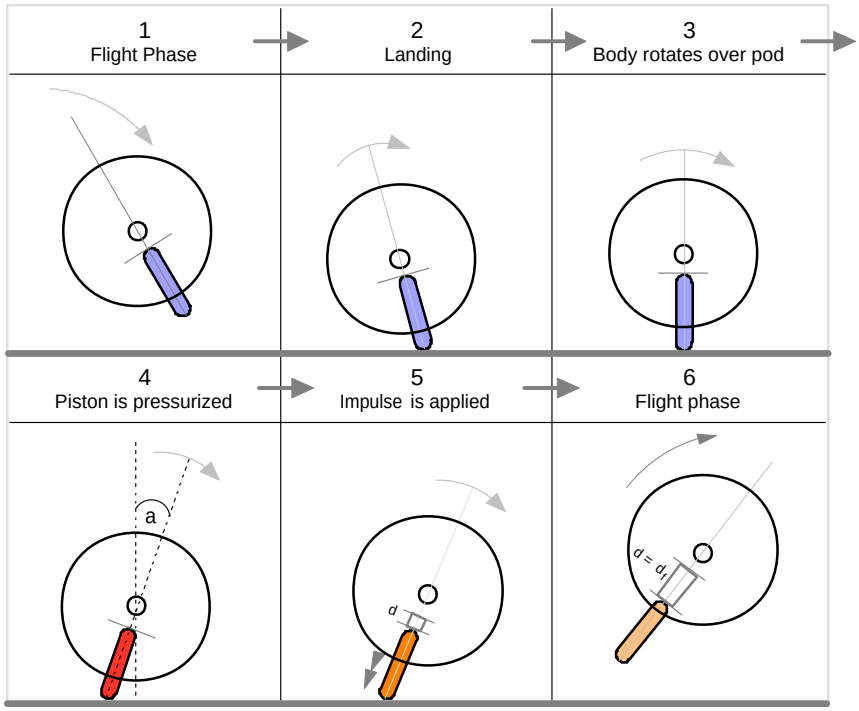


Figure 5.7: Landing (stance phase) and takeoff (flight-phase) of the robot on one leg sequence for jumping

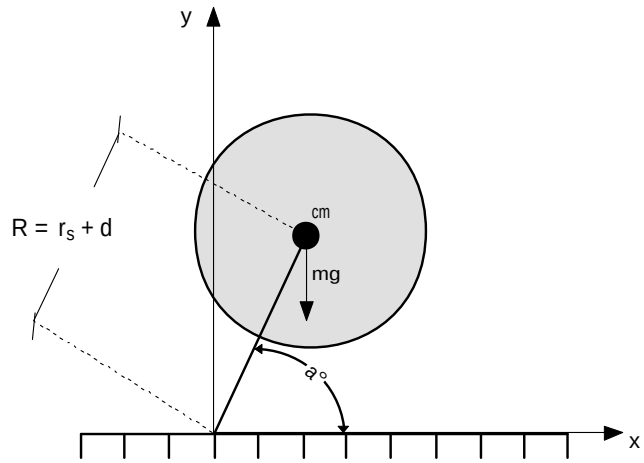


Figure 5.8: Rigid body diagram of the robot with one leg in contact with the ground plane

5.2.1 Dynamics Model

Figure 5.8 describes the basic configuration of the dynamics model. Where α ("a" in the figure) is the angle at the contact between the leg and the ground surface, R is the the radius distance between the center of the disc and the contact point. R is divided in two parts, r_s is the radius of the wheel and d is the extension distance of the leg. The disc is modeled as thin disc with mass m, and the leg is assumed to be massless.

To determine the dynamics of the system when a force is applied by the leg, the Lagrangian Equations are used. The Lagrangian equations are the following:

$$L = T - U$$

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}_i} - \frac{\partial L}{\partial \theta_i} = \tau_i$$

T is the kinetic energy and U the potential energy. The generalized coordinates are: the piston travel "d" and the angle between the the ground

plane and center of the disc with respect the contact "α" .

$$U = mg(d+R)\sin(\alpha)$$

$$T = \frac{1}{2}I\dot{\alpha}^2 + \frac{1}{2}m\dot{\alpha_i}^2(d+R)^2 + \frac{1}{2}m\dot{d}^2$$

$$L = \frac{1}{2}I\dot{\alpha}^2 + \frac{1}{2}m\dot{\alpha}^2(d+R)^2 + \frac{1}{2}m\dot{d}^2 - mg(d+R)\sin(\alpha)$$

$$\frac{\partial L}{\partial \alpha} = -mg(d+R)\cos(\alpha)$$

$$\frac{\partial L}{\partial \dot{\alpha}} = I\dot{\alpha} + m\dot{\alpha}(d+R)^2$$

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\alpha}} = I\ddot{\alpha} + m\left(\ddot{\alpha}(d+R)^2 + 2\dot{\alpha}\dot{d}(d+R)\right)$$

$$\frac{\partial L}{\partial d} = m\dot{\alpha}^2(d+R) - mg\sin(\alpha)$$

$$\frac{\partial L}{\partial \dot{d}} = m\dot{d}$$

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{d}} = m\ddot{d}$$

F is the applied force by the pressure in the cylinder.

$$I\ddot{\alpha} + m\left(\ddot{\alpha}(d+R)^2 + 2\dot{\alpha}\dot{d}(d+R)\right) + mg(d+R)\cos(\alpha) = 0$$

$$m\ddot{d} - m\dot{\alpha}^2(d+R) + mg\sin(\alpha) = F$$

Solving for $\ddot{\alpha}$ and $\ddot{d}$

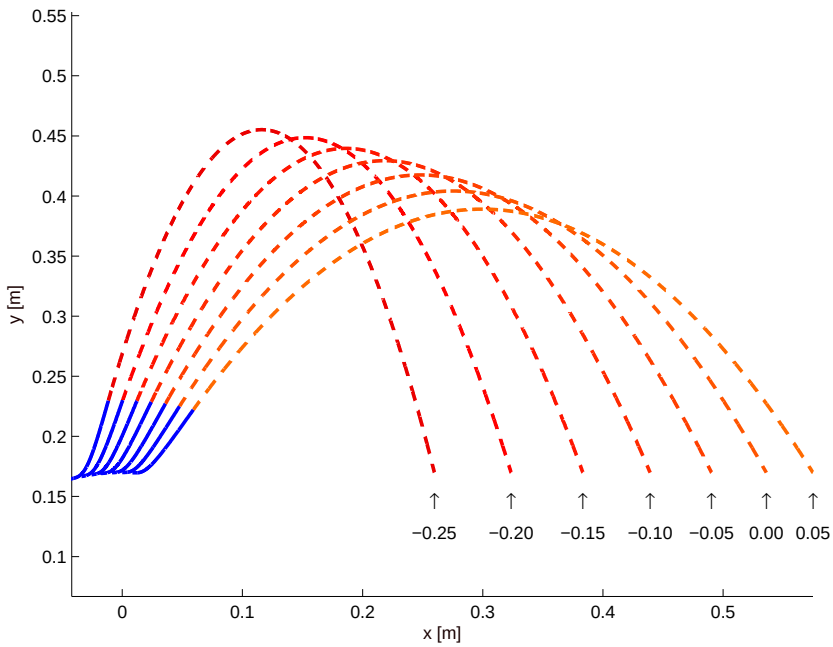


Figure 5.9: Family of trajectories of the center of the robot under different firing angles. Each trajectory has a stance phase *(solid line)* and a flight phase *(dotted line)*. The number below each trajectory is the firing angle α_f.

$$\ddot{\alpha} = \frac{-mg(d+R)\cos(\alpha) - 2m\dot{\alpha}\dot{d}(d+R)}{I + m(d+R)^2}$$

$$\ddot{d} = \frac{F + m\dot{\alpha}^2(d+R) - mg\sin(\alpha)}{m}$$

5.2.2 Model Analysis

For integrating the equations, different initial angular velocities ($\dot{\alpha}$) and firing angles α_f with respect the normal of the ground are tested. If the piston reaches the maximum displacement ($0.06m$) the robot is assumed to be in flight phase and the departing angular and linear velocities are used to simulate a rigid body with no constraints (ballistic motion). The force applied was obtained from the pressure dynamics model from Section 4.3.

Figure 5.9 shows a family of trajectories of the center of the wheel under different firing angles (α_f). The contact phase is represented by the solid lines and the flight phase by the dotted lines. The approaching angular velocity of the wheel was set at $\dot{\alpha} = -5.0[rad/s]$. The firing angles range from $\alpha|f : -0.25[rad]$ to $0.05[rad]$. The plot clearly shows that the later the leg is fired (larger α_f) the greater is the distance travelled in the horizontal direction and least in the vertical direction.

The ability to predict the trajectory of the robot after it has been fired at a certain α_f with a determined initial angular velocity can be very powerful for planing purposes. The main drawback of this model is that it assumes there is no slipping of the foot, assumption that is wrong since the foot slips as seen in the simulator or real robot. It is important as future work to measure the effect of the slipping to decide if it is relevant enough to be modeled.

5.2.3 Jumping controller

To test the jumping concept a preliminary jumping controller was implemented for the 5-legged robot. The strategy consisted in accelerating using the first three legs using the running mode presented at Section 5.1 to acquire horizontal momentum. Once the third leg is fired the controller is switched into the jumping control mentioned above using the fourth leg to jump. The transition occurs very naturally because the actuation is very discrete between legs, therefore switching controllers between firings does not disturb the behavior of the robot.

A problem found was that the horizontal velocity prior to the jump was

not repeatable. The most probable causes are changes in the friction between the foot and the floor, and non-predictable behavior of the air cylinders. As a consequence, the initial $\dot{\alpha}$ of the jumping leg was affected as well, therefore using a constant α_f generates unpredictable flight trajectories. To overcome this problem, the dynamic model presented before was used to find a relation between the α_f and the initial $\dot{\alpha}$ that generates a more uniform trajectory. The objective is to adjust the α_f according to initial $\dot{\alpha}$ to achieve a certain goal regardless the approaching speed. The goal can be to reach always the same height, or reach always the same distance, if possible.

The controller implemented was designed to jump over a constant horizontal distance of 1 meter regardless the approaching speed. If the robot approaches at a high speed then most of the propulsion is used to gain height by using a small α_f. If the robot approaches slow then a large α_f is used to direct most of the power horizontally and reach the target proposed.

To search for the relation between the initial $\dot{\alpha}$ and the α_f for our goal, the model was initialized with different initial $\dot{\alpha}$'s (in the expected range seen from the behavior of the robot). For each initial $\dot{\alpha}$ the α_f was varied and the jump was simulated. The α_f was adjusted till it generated a jump that covered the 1 meter distance within a small error margin. This was done for 5 different initial $\dot{\alpha}$'s, and the results proved that the relation was linear. The equation of the relation is,

$$\alpha_f = A\dot{\alpha}_i + B$$

The values $A = -0.1$ and $B = 0.46$ were found using the model as described above.

Figure 5.10 shows the experimental setup, and Figure 5.11 the results of the robot jumping over a cardboard box with a squared profile of $21x21cm$ without touching it. The setup of the experiment is detailed in Figure 5.10. The left plot is the position of the center of the wheel, and the right plot shows the valve signals, both plots present the complete jumping sequence from rest (note that the x-scale differs from the y-scale in the position plot). The same experiment is showed in the video Obstacle Jumping [16] using

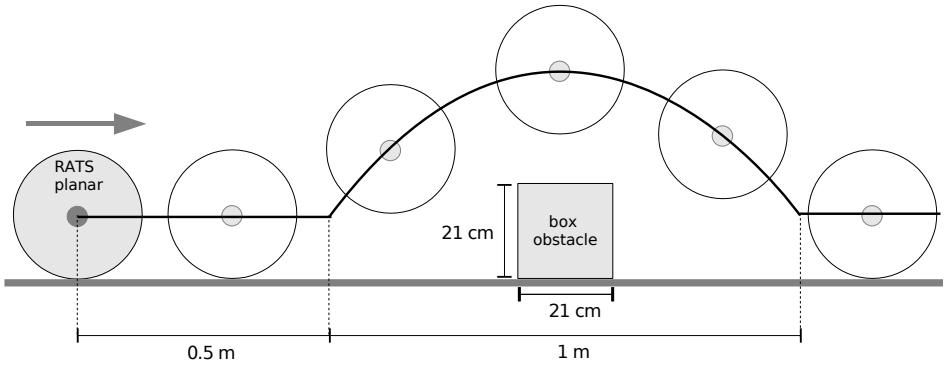


Figure 5.10: Experiment setup diagram. The robot starts at rest, accelerates in 0.5 m, and then jumps over a $21x21cm$ profile box without touching it.

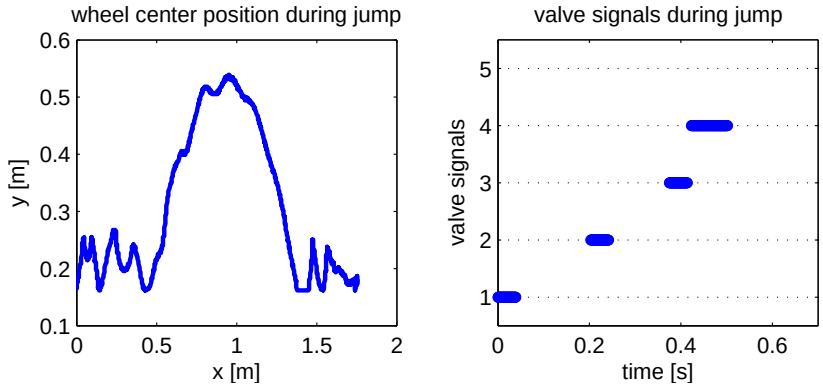


Figure 5.11: *Left plot:* Position of the center of the wheel during the jump. *Right plot:* Valve signals versus time during the jump. The y-axis labels represent the valve number. The marked zones indicate the valve was energized at that time.

different obstacles, being the second obstacle presented the $21cm$ side square box. The robot uses around 0.5 meters to accelerate horizontally using three legs sequentially, after that it uses leg 4 to jump. It can be seen that the valve opening time of the valve 4 is double the time of the others, this is given that we are trying to get the maximum power output from that leg for the jump by keeping the valve open till it reaches the hardstop, therefore injecting the maximum energy possible.

5.3 Discussion and Future Work

The key complication regarding the controllability of RATS is controlling the angle between a certain leg and the ground before it is fired. Its underactuated wheel makes the control of the robot a challenge since there is no way of directly controlling the angle between a leg and the normal of the surface. Other robots like Raiberts single-legged hoppers [20] have a lightweight leg that can be servoed to arbitrary angles when the robot is in the air. In those robots, since the body is much heavier, the motion of the leg does not disturb much the pose of the body. Also, they use the same actuator to correct the pose of the body when the leg is in contact with the ground. Conversely, in RATS the body itself is the 5 legs, and they are all fixed into the center wheel. Fortunately, the fact the wheel can roll helps in the positioning of the legs with respect the ground. The circular shape and the radial location of the pistons of RATS enable many of the control solutions proposed.

Chapter 6

Reinforcement Learning Control Techniques

This chapter focuses mainly in trying to find control strategies for the planar robot using reinforcement learning techniques to perform tasks as accelerating from rest or jumping over obstacles. This approach has advantages over traditional control schemes in the sense that it can automatically adapt to new environments without having to be tuned.

Several algorithms were tried to allow the robot to learn from the environment with no prior knowledge of it. Successful results were found to accelerate from rest using this techniques in simulation and on the real robot.

In this chapter the algorithms used are introduced first, then the simulation and robot results are presented, finally, the results are compared to those of the deterministic controller from Chapter 5 in the last section.

6.1 Reinforcement Learning

Reinforcement Learning (RL) refers to a class of problems in machine learning which postulate an agent exploring an environment in which the agent perceives its current state and takes actions. The environment, in return, provides a reward (*see* Figure 6.1). Reinforcement Learning algorithms attempt to find a policy for maximizing the discounted cumulative reward for

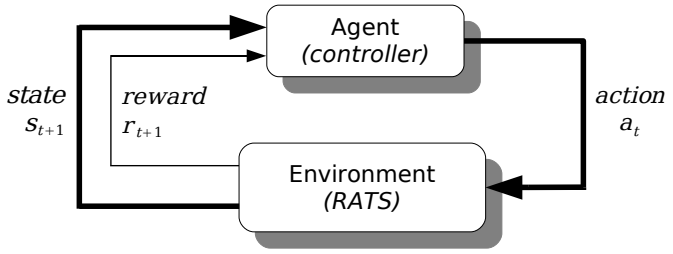


Figure 6.1: The reinforcement learning model. The agent commands an action, and then retrieves the resultant reward and state from the environment

the agent over the course of the problem [23] [8] [1].

There are three fundamental parts of a reinforcement learning problem: the environment, the reward function, and the value function. RL systems learn a mapping from situations to actions by trial-and-error interactions with a dynamic environment. For the control tasks at hand, the feedback from the environment is obtained through sensors in the robot as optical encoders and inertial sensors that measure part of the state of the robot. The "goal" of the RL system is then defined using the concept of a reward function, which generates the rewards whose cumulative values the agent seeks to maximize. In other words, there exists a mapping from state/action pairs to rewards; after performing an action in a given state the RL agent will receive some reward in the form of a scalar value. In our case, the reward function will be formulated accordingly to the objective task. If the task is to accelerate from rest the reward will be proportional to the angular velocity, which we try to maximize.

The third element in the Reinforcement Learning process is to address the issue of how the agent learns to choose "good" actions, or even how we might measure the utility of an action. For this we define two terms - policy and value. A policy determines which action should be performed in each state; a policy is a mapping from states to actions. The value of a state is defined as the sum of the rewards received when starting in that state and following a fixed policy to a terminal state. The value function therefore is a mapping from states to state values and can be approximated using a

variety of Reinforcement Learning techniques that can be used to perform this task.

The environment of RATS is retrieved by acquiring the state of the robot by using the available sensors, which include the wheel, radius and pivot angles, and the angular rate of the wheel and its accelerations. The action space is discrete and well-defined. The actions consist of firing one of the five pistons (1-5) or none (0) at each, thereby providing thrust to any one of the five legs of the robot or doing nothing at each step.

Regarding which RL technique to use, we wish to find an adequate learning algorithm that is tailored to work best for the definition of the environment, state/action space and reward function for RATS. Four different RL were considered: SARSA (Single-Step), Q-Learning (Single-Step), SARSA (Lambda - Eligibility Traces) and Q-Learning (Lambda - Eligibility traces.) Each of these algorithms was implemented in C and tested with the planar RATS simulator. One of the solutions was also tested in the planar robot to proof the concept. The algorithms and the experimental results are explained in detail at the next sections.

6.2 Methodology

6.2.1 Proposed method and algorithms description

The objective is to allow the RL algorithm to search for a suitable policy that accelerates the robot from rest fast to the running mode. We wish to summarize here the environment, actions, reward function and goal of the Reinforcement Learning problem that we are trying to formulate for the control of RATS:

1. **Environment**: Feedback from the environment is obtained through the optical encoders that measure the wheel angle, and gyroscopes that measure the angular velocity of the robot.

2. **Goal**: Maximize the angular velocity of the robot starting from rest.

3. **Action**: Open one of the piston valves (1-5) or none (0), thereby providing thrust to only one of the five legs of the robot or doing nothing at a particular time instant.

4. **Reward**: Positive reward if it produces an angular velocity greater than a certain threshold, in a particular direction (clockwise or anti-clockwise) and a negative reward in all other cases.

Having briefly formulated the RL framework for RATS, popular RL techniques are analyzed for to this particular problem. The algorithms taken in consideration are SARSA and Q-Learning algorithms. They are both popular learning techniques and have been widely used in many control tasks. SARSA and Q-Learning are both similar algorithms except that Q-Learning is off-policy (exploratory moves do not affect the policy.) SARSA is a variant of Q-Learning that is based on policy-iteration [23] [10].

1. SARSA (One Step TD Control)

2. Q-Learning (One Step off-policy TD Control)

3. SARSA (Lambda - Eligibility Traces)

4. Q-Learning (Lambda - Eligibility Traces)

We explored the SARSA and Q-Learning techniques augmenting them with eligibility traces to obtain more general methods that may learn more efficiently. An eligibility trace (given by the lambda parameter) is a temporary record of the occurrence of an event, such as the visiting of a state or the taking of an action. The trace marks the memory parameters associated with the event as eligible for undergoing learning changes. When a Temporal Difference (TD) error occurs, only the eligible states or actions are assigned credit or blame for the error. Thus, eligibility traces help bridge the gap between events and training information. Like TD methods themselves, eligibility traces are a basic mechanism for temporal credit assignment.

In the next few sections we shall illustrate the algorithmic steps for each method described above.

6.2.2 SARSA Algorithm

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., using ϵ-greedy)
  Repeat (for each step of episode):
    Take action a, observer r, s'
    Choose a' from s' using policy derived from Q (e.g., using ϵ-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma Q(s',a') - Q(s,a)]$
    $s \leftarrow s'; a \leftarrow a'$
  until s is terminal

In the algorithm ϵ-greedy stands for a greedy policy search where it chooses the previous best action each time except during ϵ percent of the time where it chooses a new random action.

6.2.3 Q-Learning One-Step Off-Policy Algorithm

One of the most important breakthroughs in Reinforcement Learning was the development of an off-policy TD control algorithm known as Q-learning (Watkins, 1989.) Q-Learning is perhaps the most popular of reinforcement techniques that are being used till date. The One-step Q-Learning update rule is given by:

$$\mathbf{Q}(s_t,a_t) \leftarrow \mathbf{Q}(s_t,a_t) + \alpha [r_{t+1} + \gamma max_a \mathbf{q}(s_{t+1},a) - \mathbf{Q}(s_t,a_t)] \tag{6.1}$$

6.2.4 SARSA (Lambda)

In order to change the One-step version of SARSA to the one with eligibility traces, we need to define a trace for each state/action pair which we denote as $e(s,a)$. The update equation becomes:

$\mathbf{Q}_{t+1}(s,a) = \mathbf{Q}_t(s,a) + \alpha\delta_t e_t(s,a)$, for all s,a

where $\delta_t = r_{t+1} + \gamma \mathbf{Q}_t(s_{t+1},a_{t+1}) - \mathbf{Q}_t(s_t,a_t)$

and

$$e_t(s,a) = \left\{ \begin{array}{ccc} \gamma\lambda e^{t-1}(s,a) + 1 & \text{if} & s = s_t \text{ and } a = a_t; \\ \gamma\lambda e_{t-1}(s,a) & \text{otherwise} & \end{array} \right\}$$

6.2.5 Q-Learning (Lambda)

The update rule for Q-Learning with eligibility traces (a combination of the Q-Learning One-step algorithm with the eligibility traces inclusion described in the previous section) is given by:

$$\mathbf{Q}_{t+1}(s,a) = \mathbf{Q}_t(s,a) + \alpha\delta_t e_t(s,a), \text{ for all } s, a$$

where $\delta_t = r_{t+1} + \gamma max_{a'} \mathbf{Q}_t\left(s_{t+1}, a'\right) - \mathbf{Q}_t(s_t, a_t)$

and

$$e_t(s,a) = I_{ss_t}\dot{I}_{aa_t} + \left\{ \begin{array}{ccc} \gamma\lambda e^{t-1}(s,a) + 1 & \text{if} & \mathbf{Q}_{t-1}(s_t,a_t) = max_a \mathbf{Q}_{t-1}(s_t,a_t); \\ 0 & \text{else} & \end{array} \right\}$$

where I_{xy} is an identity indicator function with is 1 if $x = y$ and 0 otherwise. We shall discuss the implementation details of all four algorithms along with the results in the next section.

6.3 Experimental Results

The algorithms were tested on the planar simulator and then some of them on the real robot. The four learning algorithms discussed in the previous section were coded in C on a independent library that could be ran either in the simulator or robot.

6.3.1 Simulation Results

In this section the experimental results of the simulation trials are presented. The state-space of RATS is comprised by the wheel angle, wheel angle rate, height angle and height angle rate. All these are continuous variables and hence our state space is a continuous space in four dimensions. The state-

Parameter	Discrete Steps	Range
Wheel Angle [rad]	31	-π to π
Wheel Angle Rate [rad/s]	31	(-10.1) to (21.0)
Height Angle [rad]	2	(-2.0) to (0.1)
Height Angle Rate [rad/s]	2	(-2.0) to (2.0)

Table 6.1: State-space discretization parameters. There are four states as shown above that are discretized into fixed set of values within a specified range.

Algorithm	e-greedy Disc.	γ	α	λ
SARSA	0.99	0.3	1.0	–
Q-Learning	0.99	0.3	1.0	- -
SARSA (λ)	0.95	0.3	0.8	0.9
Q-Learning (λ)	0.95	0.3	0.8	0.9

Table 6.2: This table summarizes the parameter values chosen for each algorithm tested. The e-greedy discount (given by epsilon) decides the trade-off between previous and random choices. We decrease the discount parameter to reduce choosing random actions as the iterations increase. The update parameters (α - learning rate, γ - discount factor and λ - traceability) are estimated empirically on a trial-and-error basis to give adequate outputs.

space was discretized into a table to use the methods described above. Each state was discretized into a certain number of steps as summarized in Table 6.1. The number of discrete steps for each state and the range of values are illustrated in the table.

The parameters for all four algorithms that were chosen after trying several different combinations are summarized in Table 6.2. To show how the learning algorithms improve with respect the number of episodes, three snapshots of the learning process are presented in Figure 6.2 using the One-step SARSA algorithm.

The top plot of Figure 6.2 compares the angle of the wheel versus the step at different episodes. With one episode *(dot-point line)*, the wheel barely moves. Since the objective of the problem is to make the wheel rotate as

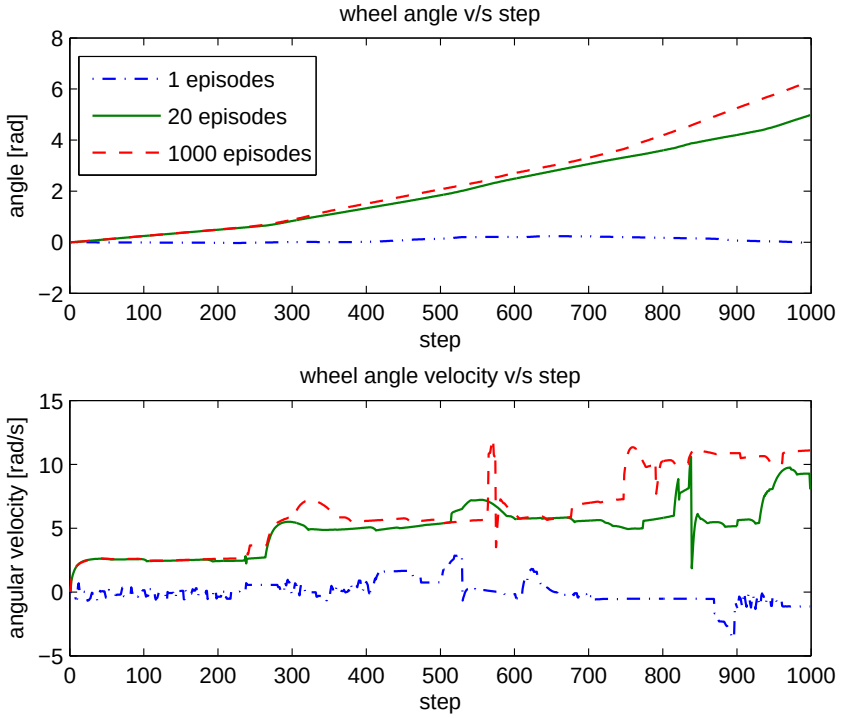


Figure 6.2: Three snapshots of the planar RATS learning to run in simulation from rest using the one-step SARSA algorithm after 1000, 20 and 1 episodes. The top plot is the angle of the wheel versus time and the bottom plot is the wheel angle rate versus time. The figure shows the progress of the learning process, by trying the learned policy after 1000, 20 and 1 episodes respectively.

fast as possible in one direction, this solution is not satisfactory. If we now allow the algorithm to run for 20 episodes, the solution presented improves dramatically *(solid line)*. Now it can be seen that the wheel rotates almost 5 radians in 1 second. Finally, if it runs till the total reward per episodes converges, the result is even better *(dotted line)*. The bottom plot from Figure 6.2, shows the angular velocity of the wheel, which is the value to optimize.

Figure 6.3 shows which actions were performed at each step. Each of the three plots were generated after a different number of episodes (*top*: after 1 episode, *middle*: 20 episodes, *bottom*: 1000 episodes). The Y axis is the number of the leg that was fired, where 0 means no leg was fired, and numbers 1 to 5 indicate which of the five legs was fired at each time step.

It is interesting to note that with one episode (the uppermost plot) the actions chosen look randomly distributed and do not make much sense. On the other hand, the bottom plot shows the best result obtained. The plot shows a more ordered pattern, where actions are grouped tightly to ensure a good thrust. The actions that have a circle around them actually represent the legs that are doing most of the work, since they are against the ground in that step. The firings outside the circles are legs fired in the air, probably to improve the behavior given that they might unbalance the wheel and help the wheel to rotate in the correct direction.

In order to compare the performance of the four algorithms, the average angular velocity of the wheel through a complete episode was taken as the benchmark. Figure 6.4-left shows the average angular velocity for each episode for the one-step SARSA and Q learning algorithms during the whole learning process (from episode 0). Figure 6.4-right shows the same results, but for the Q-learning and SARSA algorithms with eligibility traces.

The most relevant aspect of this comparison is that the algorithms augmented with eligibility traces converge much faster than their one-step counterparts. The one-step algorithms take almost 500 episodes to converge, while the ones with eligibility traces have satisfactory results with only 50 episodes (note that the X- axis scales of the plot are different). Although the algorithms with eligibility traces converge faster, they do have some

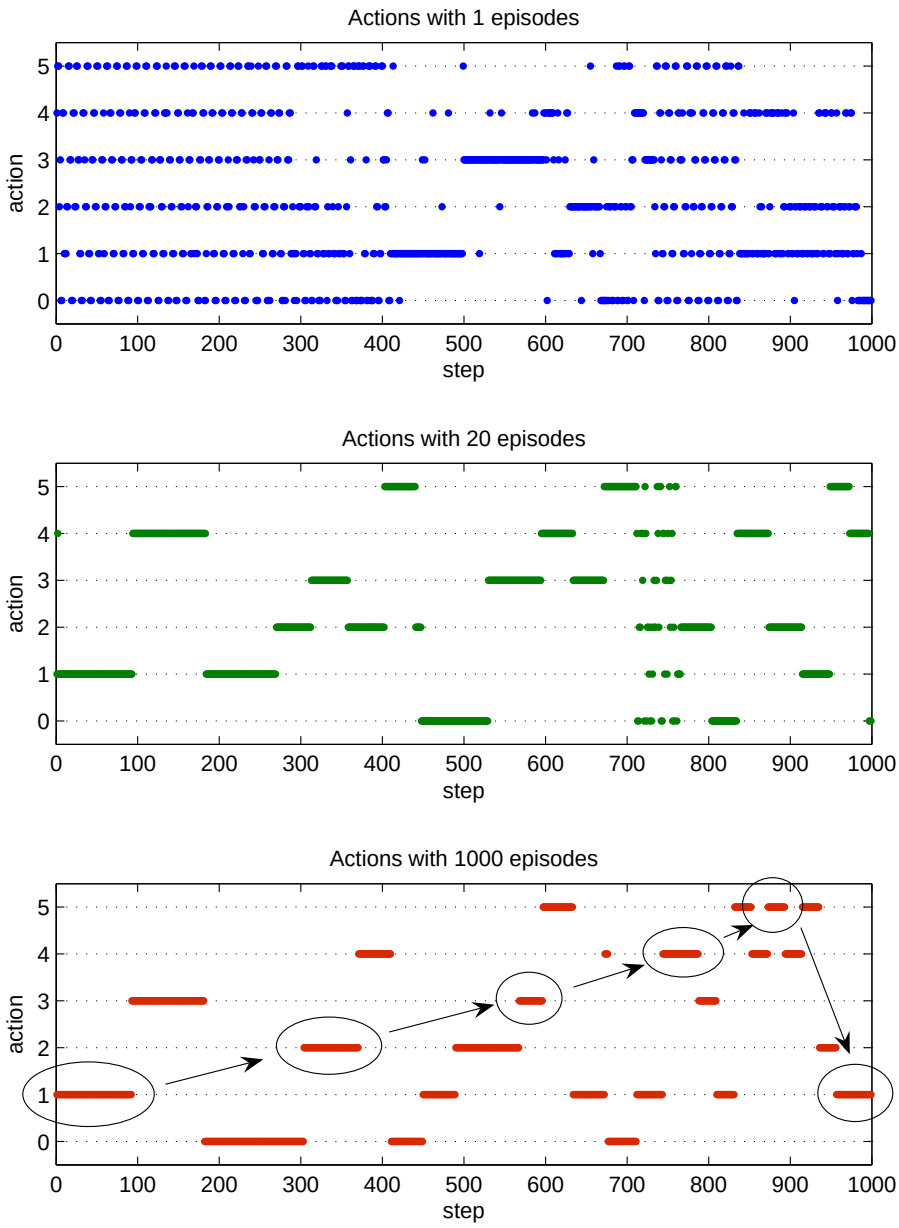


Figure 6.3: The actions performed at each step using the one-step SARSA algorithm after 1, 20, and 1000 versus time. The *y-axis* label represents the leg number, the marked zones show when each legs was fired

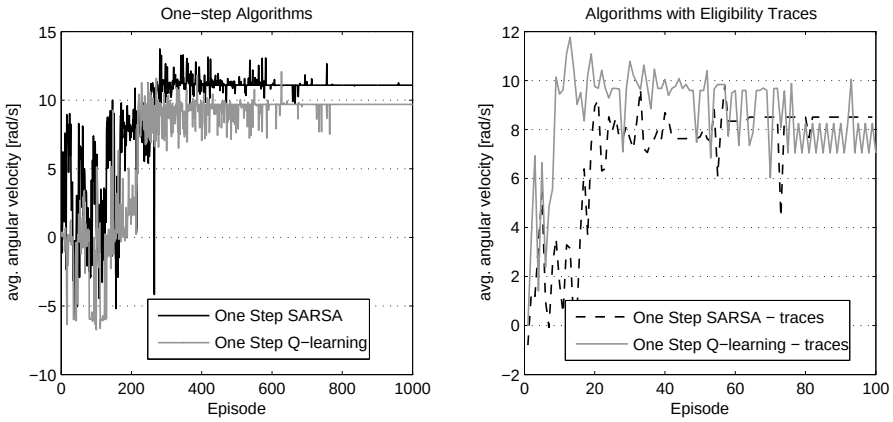


Figure 6.4: Average angular velocity for each episode run for all four algorithms. The *left plot* compares the results between the one-step SARSA and the Q-Learning algorithms. The *right* plot shows the results from the same algorithms but enhanced with eligibility traces

drawbacks - the important one being that their final scores are not as good as the one-step methods. Also, it is important to note that the amount of computation required per episode is much higher than one-step updates.

Figure 6.5 shows the angle and angular velocity of the wheel using the final learned policy of each of the four methods. Again, the results are pretty similar for all four, however, there is small edge in performance for the one-step algorithms. Also, the two algorithms based on Q-learning converged to better final scores than the SARSA based.

Another interesting result was to add a box obstacle (21x21cm) in the path of the robot as in Section 5.2.3. The reward function was now proportional to the radius angular rate instead that of the wheel angle rate, and there was a strong final reward if the robot could climb the box. The box was placed at approximately 50cm from the robot. The solution shown in Learning to Climb onto a Step [14] video was obtained using one-step Q-learning, and took 500 episodes to converge to a successful solution. It is interesting how it uses the same concept as the empirically found controller. The robot accelerates horizontally and then uses one leg to provide the vertical thrust.

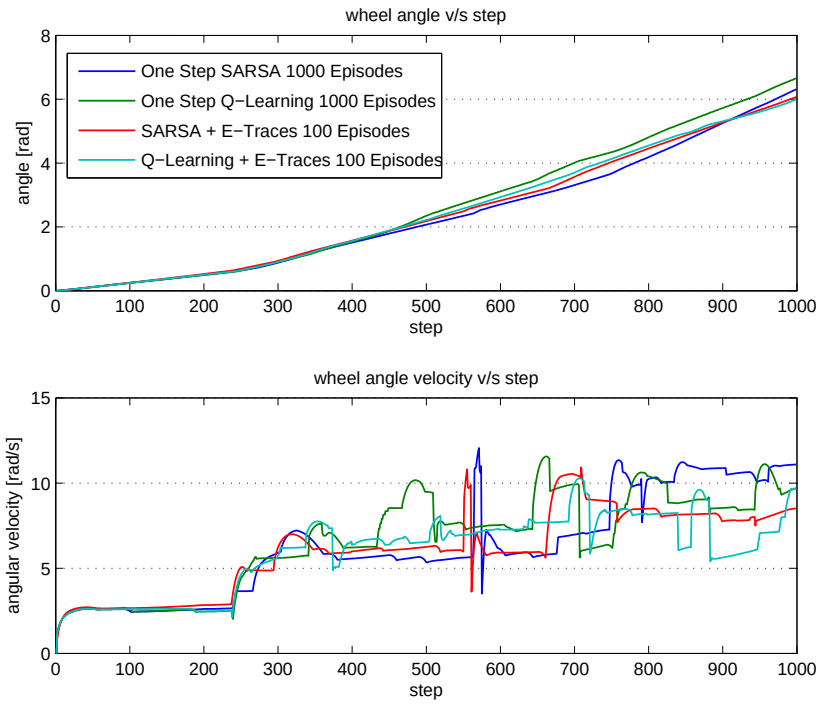


Figure 6.5: The angle and angular velocity of the wheel using the final learned policy of each of the four algorithms.

Parameter	Discrete Steps	Range
Wheel Angle [rad]	31	-π to π
Wheel Angle Rate [rad/s]	31	(-15.1) to (21.0)

Table 6.3: State-space discretization parameters for use in the controller of the planar RATS.

The analysis we have provided above was drawn from our particular results, which may have some bias based on the parameters we chose for each method. However, the conclusion is that all four methods are robust, and prove that reinforcement learning is viable solution to control complex systems such as the one presented in this work. The eligibility traces based methods proved to learn very fast, and therefore suitable for being used in the real robot.

6.3.2 Robot Results

The first relevant consideration to test the RL algorithms in the real robot was to chose an algorithm that was feasible for being used in a real physical machine. The SARSA one-step or Q-Learning one-step algorithms need around 500 episodes of 1000 steps to learn the intended behavior. This is not practical for use in real world applications since the objective is that the robot learns "fast" its new environment so that the learning mode can be applied every time the robot is presented to a new environment.

The learning methods enhanced with eligibility traces were chosen since they are much better suited to our application since they need fewer episodes to learn the behavior.

The transition from the simulator was not a problem since the RL library was written independently from the simulator and was easily added to the robot controller. The initial concern was that the controller was not going to be able to do all the computations required in time. To reduce the computation load a two-dimensional state-space was used, comprised of the wheel angle and the wheel angular rate only as described in Table 6.3. This reduced the size of the **Q** dramatically and therefore the time to recalculate

the traces and Q values each time step. Another concern was that the original controller was designed to fire any of the 5 legs in each step, this included the legs that were pointing to the air. Although this legs might help to achieve the goal by adding small inertias, they were considered negligible given then legs are very light. Therefore, it was decided that only the leg in the *active zone* (explained in Section 5.1) could be fired, this was done to reduce the wear of the air cylinders and valves and speed up the learning process. This furthermore reduced the **Q** table from 6 different actions to just 2, which corresponded to fire or not the leg that was in the *active zone*.

During the initial tests the computer was able to handle all the additional computations of the eligibility traces without generating late pulses in the control loop. Nevertheless, the learning process was not working due that the actuation time was too small for the valves to actuate. When the learning process was started the valve of the leg in the *active zone* was receiving random pulses of 1 ms (the loop cycle time) which were too short to cause the piston to move. To solve this problem it was decided to run the RL slower so that individual actions were long enough as to cause a change in the state of the robot. The RL learning loop was finally ran at *200Hz*, increasing the minimum actuation to $5ms$.

With the modifications proposed the learning process worked fine, and the robot was able to start from rest and accelerate to a fast speed in a short time of period. The only algorithm that was tried was the SARSA with eligibility traces. Also, the algorithm was not completely repeatable every time the the learning process was started from the beginning, some times it would stall. The possible causes of this are going to be discussed after the results are presented.

Table 6.4 shows the parameters that govern the SARSA(λ) algorithm used. Figure 6.4 shows the results of the robot learning to run in 11 episodes. The figure shows only episodes 1, 4, and 11, which were chosen to be representative of the process. During episode 1 the controller has not information of the environment therefore the actuation of the valves are almost random, and the robot just inclines once and then comes back to the initial position as displayed in the figure. From episode 4 to 11 the robot learns to run

Algorithm	e-greedy Disc.	γ	α	λ
SARSA (λ)	0.95	0.3	0.8	0.9

Table 6.4: This table shows the parameters chosen for the SARSA(λ) algorithm implemented in the planar robot. After searching different values, we found the most suitable were the same values that were used in the simulation (which were our starting point)

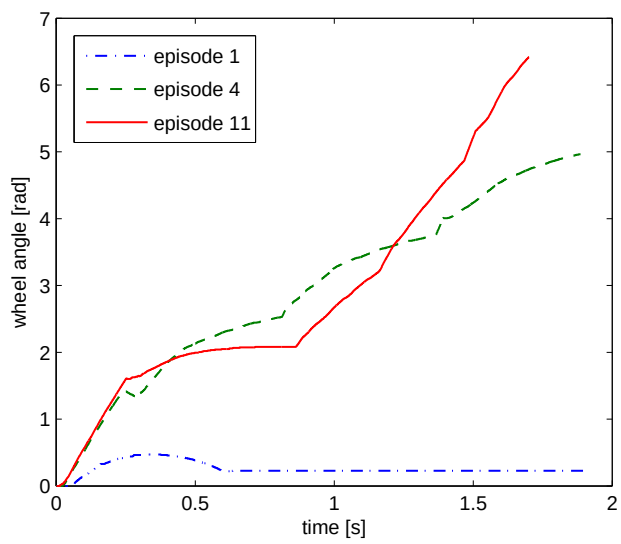


Figure 6.6: Wheel angle progress of episodes 1, 4, 11. Data acquired using the optical encoder. Episode 11 is shorter in time given there was a cut-off linear distance goal of 1.6 m. If the distance was not achieved then the robot would operate till it reached 1800 steps of 1 ms.

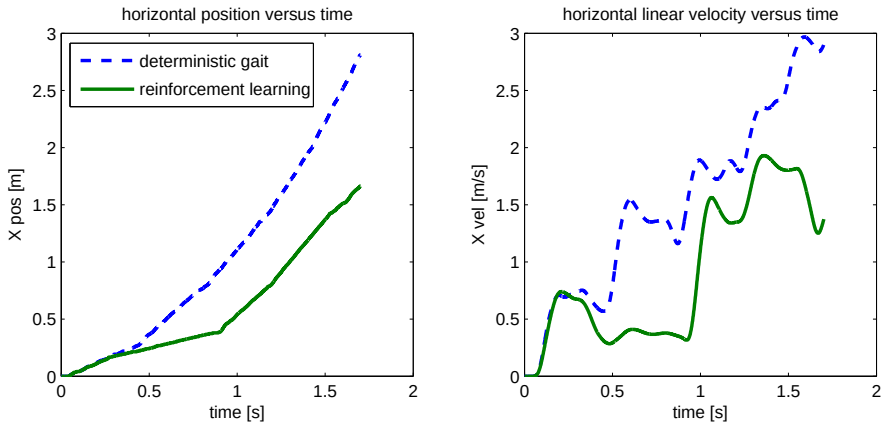


Figure 6.7: Accelerating from rest, comparison between the deterministic controller using wheel angular velocity feedback versus the RL controller. The *dotted* line represents the deterministic controller, and the *solid* line the RL controller. The *left* plot is the horizontal position versus time, and the *right* plot the horizontal linear velocity versus time.

fast, as the figure suggests, but there is not always a continuous improvement from episode to episode. Running the robot for more than 11 episodes did not improve the behavior. The video Learning to Run from Rest in 11 episodes [15] shows the 11 learning episodes of the planar robot described above.

6.4 Discussion and Future Work

An important conclusion is to compare the performance of the reinforcement learning controller with that of the deterministic controllers from Chapter 5. Figure 6.7 compares the results from both controllers, being the task to accelerate fast from rest. The tests were done using the real robot, under similar conditions. The deterministic controller used is the closed-loop controller that uses angular velocity feedback from Section 5.1.3.

From Figure 6.7 it is clear that the deterministic controller outperforms its RL counterpart. Nevertheless, the results are good enough to encourage the use of learning techniques given the great advantages they have over

traditional controls. Next the possible facts that explain the performance gap are discussed.

State-Space Representation

The RL experiment to accelerate fast from rest on the planar robot used only two state variables (wheel angle and wheel angular velocity) to represent the state. Also the discretization was very coarse, with only 31 linear steps per variable. The discretization steps and the number of state variables were left small on purpose so that the computation of the SARSA(λ) could be done in real time.

The wheel angle and its velocity probably were the most descriptive state variables of the actual state of the robot. Nevertheless, others like the pivot angle and its velocity were left aside and were very relevant to describe the state. The pivot angle gives information about the distance between the ground and the surface, since its very different to be at 10 cm from the floor than actually touching the floor. Other state variables that could have been helpful are the position of the pistons in the air cylinders. The lack of state information transforms the problem into a partially observable Markov decision problem (POMPD), for which the learning algorithms used are not designed to work with. Since the missing states are available (or can easily be added), a possible good solution would be to use function approximation methods to reduce and/or interpolate the searchable value space. Approximating the value function with continuous functions should also solve the problem of the coarse table discretization.

The implementation of the RL algorithms with function approximation is left as future work.

Chapter 7

3D Robot

Since the 3D RATS prototype is still in design and fabrication phase, a 3D simulator was implemented to commence in advance the study of the control strategies of the 12-legged system. In this chapter the robot simulator is presented first, followed by a running strategy extrapolated from the planar robot. Stabilization ideas and strategies for performing other behaviors are discussed in the final section.

7.1 Robot Simulator

The 3D robot was simulated as a free body with no tether or arm that constraints its motion. The system was simulated using the same tools as for the planar version (please see Section 4.1). The simulator was designed such that every leg was attached to the main sphere with a slider joint. The weight of each leg was assumed to be the same as the legs of the planar robot.

The body was modeled as a balanced spherical mass with constant density with a total mass of $3kg$ which was the predicted weight of the 12-legged prototype without the pistons. The radius of the sphere is the same as that of the wheel of the planar robot.

The force that each leg applies was assumed to be constant since the pneumatic system of the 12-legged will differ considerably from the one of

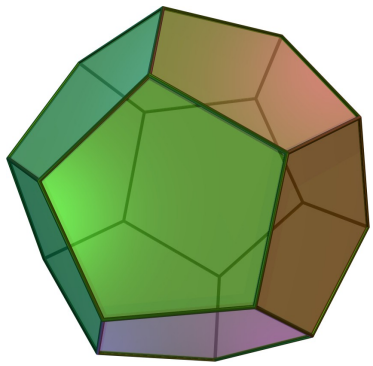


Figure 7.1: 3D RATS geometry: Dodecahedron, each leg is located at the center of each face. *Source:* Wikipedia.org

the planar robot. This is because the new prototype will use high-pressure C02 and custom made valves, whereas the planar robot used low pressure compressed air and commercially available valves. The pneumatic system of the 12-legged system has not been modeled yet.

The contacts between the feet and the surface are simulated the same way as in the planar case. The friction and slip parameters were left unchanged.

The underlying shape of the 12-legged prototype is a *dodecahedron* which is displayed in Figure 7.1, which is a *polyhedron* with 12 faces. Each leg is located at the center of each face pointing normal. The *dihedral angle*, which is the internal angle between two adjunct faces, is $\arccos(\frac{-1}{\sqrt{(5)}})$ or $116.6°$. With the *dihedral angle* the angle between the normals of two adjunct faces was calculated being of $\pi - 2\arccos(\frac{-1}{\sqrt{(5)}})$ or $63.43°$, which corresponds to the angle between any two neighbor legs in the robot.

Figure 7.2 shows a rendered view of the simulator. When the robot is at rest there are 3 legs in contact with the surface. Given the symmetry of the the *dodecahedron*, the contact points of the three legs with the surface generate an equilateral triangle on the surface plane.

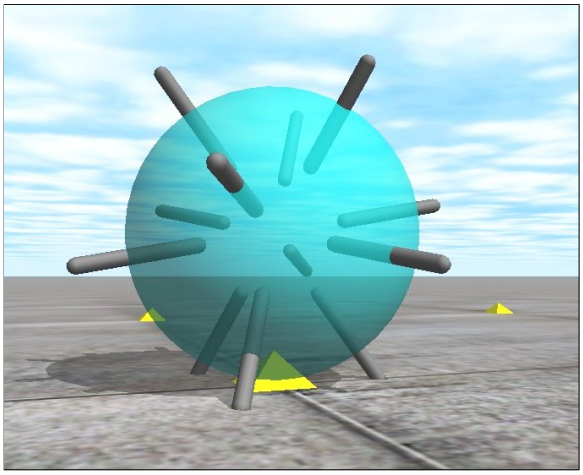


Figure 7.2: 12-legged RATS 3D simulator renderized view.

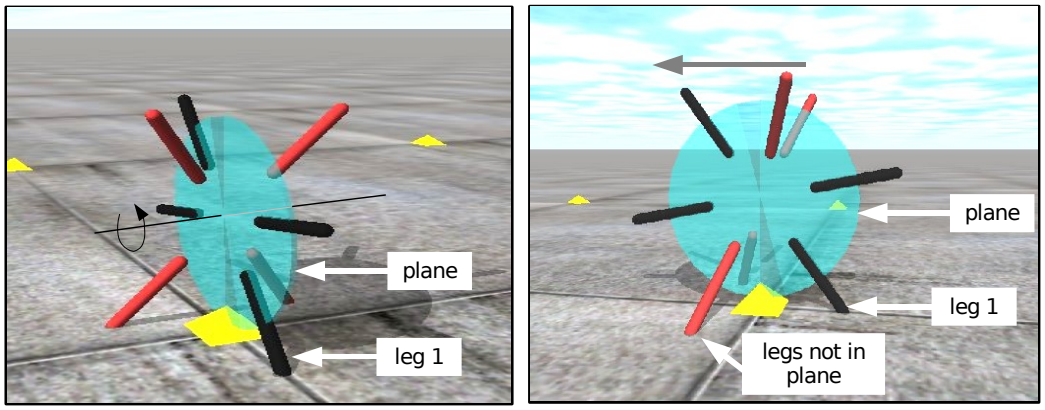
7.2 Running

In this section a running gate developed for the 3D robot is presented. This gate is based on the running strategy devised for the planar robot (please see Section 5.1).

7.2.1 Rolling Disc Concept

The basic concept is to induce the sphere to roll in a determined direction using a subset of the 12 legs of the robot. If the sphere rolls straight then the contact patch can be thought as a disc whose circumference is a *great circle* of the sphere. This disc is contained in a plane that is normal to the ground surface. Figure 7.3 shows the concept of the disc plane when aligned with one of the three legs that are in contact with the floor. Both, the left and right images show the disc plane from different view perspectives. Figure 7.3(a) shows the rotating axis which goes parallel to the surface and perpendicular to the running direction (displayed by a grey arrow in Figure 7.3(b)).

A 3D animation that gives an overview of the 12-legged simulator and the concept of the disc plane can be seen in 3D Simulator Overview Video [12]. The video clearly shows the details of the simulator, and the location



(a) (b)

Figure 7.3: Two different views of the disc plane used for running in the 3D simulator.

of the disc plane within the sphere.

7.2.2 Angle-Offset Control

From Figure 7.3, the controller operates by first firing leg 1, then the twin legs in light-shade (red legs in the video) and then the next black in the plane, and so on. The video 3D Running Gait Video [13] shows the 3D simulator accelerating from rest using the controller proposed.

The controller works by treating each of the contiguous legs that are not in the plane (light shaded legs (red) from Figure 7.3) as a single leg by firing them simultaneously as it can be seen in the video. Figure 7.4 shows the projection of all the legs used for running into the disc plane. This projection generates a wheel very similar to that of the planar robot, although it has 6 legs instead of 5. Each of the light-shaded (red) legs in the wheel represent two legs that are projected into the plane. The projection also causes the twin legs to be shorter in the disc plane, which has consequences since they can reach less far than the black ones when running. Another particular characteristic is that the angles between legs is not constant as it is clearly shown in Figure 7.4.

Once the wheel plane is correctly defined with its 6 legs, then the angle

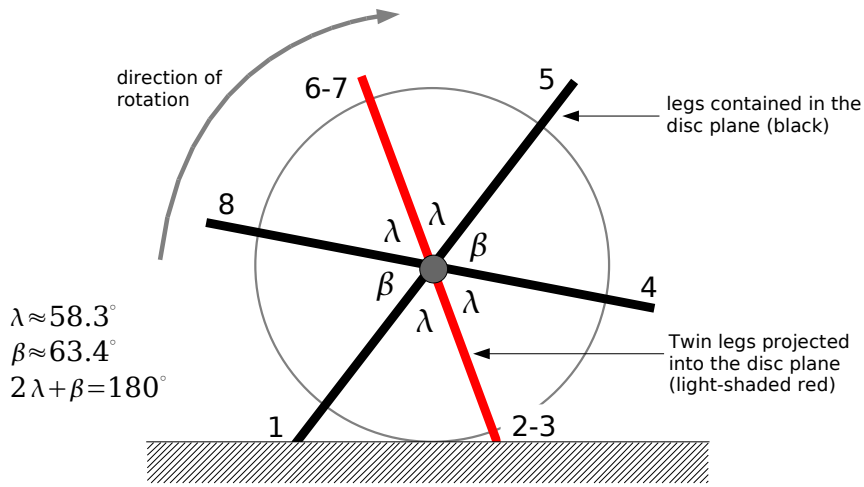


Figure 7.4: Projection of the 8 running legs into the disc plane. Each of the light-shaded legs (red) represent the projection of two legs into the disc plane.

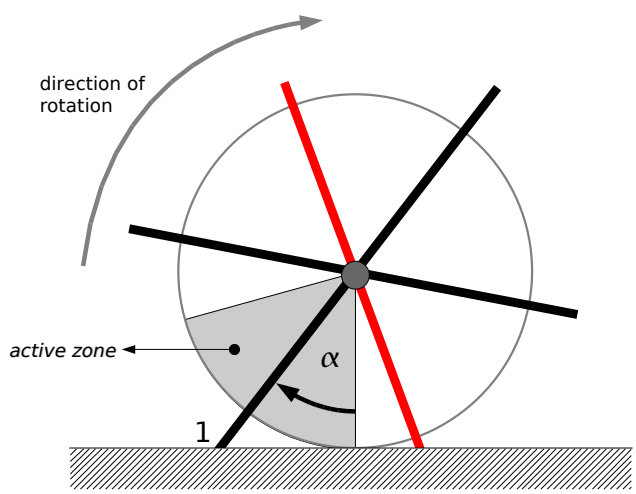


Figure 7.5: Active zone concept for running in the direction of the arrow. Any leg that enters the active zone is fired at an alpha angle called alpha-fire (α_f).

offset concept is used with an active zone and an alpha-fire angle very similar to the planar case (please refer to Section 5.1 for an explanation of the alpha-fire and active zone). Figure 7.5 shows the placement of the active zone for running by forcing the wheel to rotate clockwise. The only difference between this case and the planar robot is that the active zone aperture size changes continuously because the angle between two contiguous legs varies.

7.3 Discussion and Future Work

The controller suggested above works fine in the simulated environment because the sphere is balanced, there are no external perturbations, and all the air-cylinders behave the same. With the real robot probably non of those assumptions will be true. The sphere will be slightly unbalanced, the individual pistons will operate differently and the ground surface will not be perfectly homogeneous. All these factors will induce the robot to roll sideways and/or change its heading. This motion will tilt the the disc plane sideways causing the failure of the controller. One solution to stabilize the robot is to apply different forces to each of the twin legs when running to intend to correct the yaw and/or roll of the sphere. This solution would be able to apply a corrective action twice per turn (when the twin legs are fired).

The twins legs could also be used to steer the robot and change the heading of the robot by applying different forces to each of the twin legs.

Another interesting solution would be to extend a subset of the 4 legs that are not being used for running into the air to change the center of mass and try by that way to correct the roll angle.

Chapter 8

Discussion

This thesis addresses a variety of different topics that together attain a unifying idea which is to develop locomotion control techniques for a particular novel robot. This chapter intends to unify these different topics by comparisons and discussion.

The first section presents the conclusions of the control algorithms and simulation tools. In further sections the contributions and future work are presented.

8.1 Conclusions

8.1.1 Controls

Several control algorithms were devised to achieve running or jumping behaviors with RATS. Two main methodologies were used to develop the controllers: deterministic discrete controllers, and reinforcement learning based controllers. First, the conclusions of the deterministic and reinforcement learning controllers are presented independently, and then a comparison of both is presented.

Deterministic Controller

The deterministic controls solutions (see Chapter 5), which are largely based on intuition, proved to be very successful to conduct the robot to perform the intended behaviors. The most advanced gait developed, which uses the wheel angular velocity to adapt the firing angle based on a non-linear function, was able to command the robot from rest to a fast and smooth running mode automatically. The robot was able to accelerate from 0 to $3.2m/s$ in less than 3 seconds. Its closed-loop nature added robustness, by allowing the robot to accept perturbations while accelerating or running steadily.

The jumping controller, which uses the running gait to accelerate horizontally before take-off, was also very successful as the results show. By varying the firing angle of the last leg to fire different trajectories could be made. The model to predict the trajectory and ground clearance was very helpful to develop the jumping gait. The robot was able to successfully jump over a square box of side $21cm$ without touching it.

Reinforcement Learning Controller

The reinforcement learning algorithms tested were used to learn from the environment with no previous knowledge of it. This has important advantages such that it can adapt to new environments without external help. Using SARSA(λ) the real planar robot was able to learn how to accelerate from rest in 11 short trials. The results were not as good as those of the deterministic controller. Possible causes are a limited and coarse tabular space-state representation.

Nevertheless, the results were good enough to encourage efforts in this research direction since the advantages can be many, particularly if the robot is planned to be used in autonomous mode.

Comparison Table

Table 8.1 presents a brief comparison of the characteristics of the deterministic and reinforcement learning methods implemented in the planar robot.

	Performance	*Robustness*	*Computation & Sensing Required*
Deter-ministic	Fast acceleration, and smooth running. Speed after 1.6 seconds: 2.9 m/s (closed-loop). Successful jumps over a 21cm sides square box.	Good for the running mode with closed-loop since it can accept perturbations during acceleration and steady running.	Very low computation required, only two sensors were used for the running and jumping strategies.
ML	Slower acceleration, coarser motion. Speed after 1.6 seconds: 1.4 m/s (SARSA(lambda)). Successful jump over box in simulation, not tried in the real robot	Good against perturbations if the policy generalizes well. Should be excellent to learn new environments. No experiments were made to prove this.	High computation required for the eligibility traces algorithm. Only two sensors were used, but many more might be needed to improve performance.

Table 8.1: Comparison of the main characteristics of the deterministic and the reinforcement learning controller methods.

8.1.2 Simulation

The simulator, developed in ODE and visualized in OpenGL, was fast and accurate enough to assist in the development of the control strategies presented in this work. The addition of the pressure model improved the accuracy by modeling the pressure dynamics in the chamber. The simulator was particularly helpful to develop the reinforcement learning controllers since it can run much faster than the real robot (thousands of times).

8.2 Contributions

In this work novel deterministic gaits were created that allow the planar RATS robot to accelerate from rest and run smoothly, and also to climb or jump over obstacles. The running gaits were extrapolated to the 3D case and were proven in simulation to be possible candidates for controlling the spherical robot.

Besides the deterministic controllers, machine learning techniques were implemented proving they are able to control a robot like the planar RATS with no previous knowledge of the robot or environment.

An accurate simulator of the planar robot was developed, including a detailed model of the pressure dynamics of the air cylinders. A less detailed 3D simulator was also implemented. This tools should be of great utility for upcoming work.

8.3 Future Work

There is still an extensive amount of work to be done in the controls of RATS. In this section the future work is divided into two parts: (1) possible changes/improvements that could be done to the design of the robot, (2) future work for the control algorithms.

8.3.1 Design

The mechanical design and sensing of the robot have a very important role on the controllability of the robot. Therefore, even small changes can have an important impact on its performance.

The actual air cylinders use a spring to return the piston, which in some circumstances is too weak to retract the piston as desired. Alternatively, another solution would be to have a double acting air cylinder that can actively extend or retract the piston using a second three way valve. Additionally, if position and pressure sensors are installed in the air-cylinders, interesting controls approaches could be implemented such as using the air chamber as an air spring to store energy, and/or control the piston position before landing.

Another addition that would be welcomed, is to install pressure contact switches to the feet tips in order to know when the foot is contact with the ground surface. The information from such sensors could be used to determine transitions within the control algorithm.

Adding range sensors in the body of the robot would be useful to detect the ground plane besides using a IMU. A GPS could be used for localization.

8.3.2 Controls

The planar robot is an excellent test platform to keep trying new control techniques. A list of suggested future work is listed next:

- Development of an accurate running speed control by actuating over the firing angle (α_f) and valve opening time (t_v).
- Study the energy efficiency problem.
- Use planning methods for jumping that use sensing to detect an obstacle and then decide the correct time for take-off while running.
- In the simulator, incorporate the geometry of the foot and calibrate the frictions to improve accuracy.

Many areas in the controls should continue to be developed for the planar robot, nevertheless, the most exciting aspect is to start to test and develop control strategies for the 3D robot. These would include stabilizing solutions for the proposed running gait, and devise a jumping gait that could be based on the one of the planar robot.

Another interesting approach that needs further development is the use of machine learning techniques. In this area better techniques should be tested such as function approximation with an extended state-space using extra sensors. A disctint approach would be to use reinforcement learning or other learning technique to learn higher order parameters such as the firing angle instead of which legs to fire in each time step, reducing the problem into a simpler and more structured one.

Bibliography

[1] Andrew G. Barto, Richard S. Sutton, and Christopher Watkins. Learning and sequential decision making. In Michael Gabriel and John Moore, editors, *Learning and computational neuroscience : foundations of adaptive networks*. M.I.T. Press, Cambridge, Mass, 1990.

[2] Shourov Bhattacharya and Sunil Kumar Agrawal. Design, experiments and motion planning of a spherical rolling robot. In *ICRA*, pages 1207–1212. IEEE, 2000.

[3] Ben Brown and Garth Zeglin. The bow leg hopping robot. In *ICRA*, pages 781–786, 1998.

[4] Ben Brown and Garth Zeglin. Control of a bow leg hopping robot. In *ICRA*, pages 793–798, 1998.

[5] Joel Burkick and Paolo Fiorini. Minimalist jumping robots for celestial exploration. *I. J. Robotic Res.*, 22(7-8):653–666, 2003.

[6] Centro e. Piaggio, Antonio Bicchi, Andrea Balluchi, Domenico Prattichizzo, and Andrea Gorelli. Introducing the "SPHERICLE": an experimental testbed for research and teaching in nonholonomy, 1997. Warning: the year was guessed out of the URL.

[7] Paolo Fiorini and Joel W. Burdick. The development of hopping capabilities for small robots. *Auton. Robots*, 14(2-3):239–254, 2003.

[8] M. Harmon. Reinforcement learning: a tutorial, 1996.

[9] National Instruments. *"NI-DAQmx Help"*. 11500 N Mopac Expwy Austin, TX 78759-350, 370466f-01 edition, September 2005.

[10] Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[11] Rolf Allan Lüders. Planar robot running using the fixed-angle control video. http://www.frc.ri.cmu.edu/∼aluders/thesis/videos/RATS-5LP-running-fixed-angle-offset.wmv, 2006.

[12] Rolf Allan Lüders. 3-dimensional simulator overview animation video. http://www.frc.ri.cmu.edu/∼aluders/thesis/videos/RATS-3D-sim-overview.wmv, 2007.

[13] Rolf Allan Lüders. 3-dimensional simulator running gate animation video. http://www.frc.ri.cmu.edu/∼aluders/thesis/videos/RATS-3D-sim-running.wmv, 2007.

[14] Rolf Allan Lüders. Climbing over obstacle q-learning one-step video. http://www.frc.ri.cmu.edu/∼aluders/thesis/videos/RATS-5LP-SIM-RL-Obstacle-Climbing-II-Q-Learning.wmv, 2007.

[15] Rolf Allan Lüders. Learning to run from rest in 11 episodes. http://www.frc.ri.cmu.edu/∼aluders/thesis/videos/RATS-5LP-running-SARSAwET-1.avi, 2007.

[16] Rolf Allan Lüders. Planar robot obstacle jumping video. http://www.frc.ri.cmu.edu/∼aluders/thesis/videos/RATS-5LP-obstacle-jumping.avi, 2007.

[17] Rolf Allan Lüders. Planar robot running using the variable-angle control varied in closed-loop mode video. http://www.frc.ri.cmu.edu/∼aluders/thesis/videos/RATS-5LP-running-fast-accel-closed-loop.avi, 2007.

[18] Rolf Allan Lüders. Planar robot running using the variable-angle control varied in open-loop mode video. http://www.frc.ri.cmu.edu/∼aluders/thesis/videos/RATS-5LP-running-fast-accel-open-loop.wmv, 2007.

[19] K. Matsuoka. A mechanical model of repetitive hopping movements. *Biomechanisms*, 5:251-8, 1980.

[20] Marc H. Raibert. *Legged Robots That Balance*. MIT Press, Cambridge, MA, 1986.

[21] Edmond Riche and Yildirim Hurmuzlu. A high performance pneumatic force actuator system part 1 - nonlinear mathematical model. *Transactions ASME*, 2000.

[22] Rusell Smith. *Open Dynamics Engine v0.5 User Guide*, Frebruary 2006. www.ode.org.

[23] Richard S. Sutton and Andrew G. Barto. *Reinforcemente Learning: An Introduction*. MIT Press, 1998.

[24] Peter Weiss. Hop...hop...hopbots! *Science News*, 159(6):88, Feb. 10 2001.

[25] Garth Zeglin. *The Bow Leg Hopping Robot*. PhD thesis, Carnegie Mellon University, 1999.