



VILNIUS GEDIMINAS TECHNICAL UNIVERSITY
ELECTRONICS FACULTY
AUTOMATION DEPARTMENT

Jorge Diosdado Borrego

**DEVELOPMENT OF ADAPTIVE TRAJECTORY GENERATION METHOD
FOR A HEXAPOD ROBOT**

Final bachelor thesis

Electronic and Electrical study area
Automation study programme, state code 612H62002
Automatic Control of Technologies specialization

Vilnius, 2014

VILNIUS GEDIMINAS TECHNICAL UNIVERISTY
ELECTRONICS FACULTY
AUTOMATION DEPARTMENT

ASSERT
Department head


(signature)

Zita Savickienė

2014 y. 06 m. 10 d.

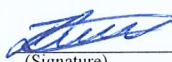
Jorge Diosdado Borrego

**DEVELOPMENT OF ADAPTIVE TRAJECTORY GENERATION METHOD
FOR A HEXAPOD ROBOT**


Final bachelor thesis

Electronic and Electrical study area
Automation study programme, state code 612H62002
Automatic Control of Technologies specialization

Supervisor dr. Tomas Luneckas
(Sci. degree, name, surname)


(Signature) 2014-06-10
(Date)

Consultant assoc. prof. dr. Bronius Karaliūnas
(Sci. degree, name, surname)


(Signature) 2014-06-10
(Date)

Consultant lect. Dalia Lukošienė
(Sci. degree, name, surname)


(Signature) 2014-06-10
(Date)

Vilnius, 2014

VILNIUS GEDIMINAS TECHNICAL UNIVERSITY
ELECTRONICS FACULTY
AUTOAMTION DEPARTMENT

Electronic and Electrical study area
Automation study programme, state code 612H62002
Automatic Control of Technologies specialization

ASSERT
Department head


(signature)

Zita Savickienė
(name, surname)

2014 y. June 6 d.
(date)

FINAL BACHELOR'S THESIS

TASK

Nr. 32.....

Vilnius

Student Jorge Diosdado Borrego

Bachelor thesis title: Development of adaptive trajectory generation method for a hexapod robot

Affirmed 2014 y. June 6 d. deanery decree No. 126el.

Bachelor thesis completion 2014 y. May 27 d.

BACHELOR THESIS TASK:

Information: 1. Hexapod robot with 18 degrees of freedom (3 DoF per leg) 2. ATmega16 microcontroller at 16MHz. 3. Circuit 74LS241. 4. Dynamixel AX-12 servomotors. 5. Power supply 12 V for servomotors and 5 V for microcontroller. 6. MATLAB simulation program.

Explanatory writing:

1. INTRODUCTION
2. ANALYTICAL PART 2.1. Stationary robots. 2.2. Locomotion overview. 2.3. Wheeled robots. 2.4. Legged robots. 2.5. Hexapod robots.
3. DESIGN PART. 3.1. Electronic circuit of the hexapod robot. 3.2. Hexapod robot kinematics. 3.3. Construction of trajectory generation function. 3.4. Implementing the parabola in the MATLAB® model. 3.5. Adaptive trajectories method. 3.6. Obstacles.
4. CONCLUSIONS. Future work.
5. REFERENCES

Final bachelor thesis consultants: doc. dr. Bronius Karaliūnas, lekt. Dalia Lukošienė.
(position, name, surname)

Supervisor.....
(signature)

Dr. Tomas Luneckas
(scientific degree and name, name, surname)

Task received

(signature)

Jorge Diosdado Borrego
(Student name, surname)

2014 y. June 6 d.
(date)

Vilnius Gediminas Technical University
Faculty of Electronics
Department of Automation

ISBN ISSN
Copies No.
Date-.....-.....

Bachelor Degree Studies **Automation** study programme Bachelor Graduation Thesis 3

Title **Development of Adaptive Trajectory Generation Method for a Hexapod Robot**
Author **Jorge Diosdado Borrego**
Academic supervisor **Dr Tomas Luneckas**

Thesis language:
English

Annotation

In this final thesis, a method for generating adaptive trajectories for a robot is proposed. For it, it is necessary to first develop the mathematical functions which robot feet will follow. The robot is a hexapod robot, which has three degrees of freedom for each leg.

Also, a revision of types of robot is carried out, including different types of locomotion.

All the checks have been carried out in a model constructed to simulate the robot, allowing a fast and effective correction of mistakes.

The work consists in four sections: Introduction, review of robotics, experiments and conclusions.

Work size: 51 pages of text, 48 figures, 2 tables.

Keywords: Hexapod robot, trajectory generation, robot simulation, legged robot, kinematics

(the document of Declaration of Authorship in the Final Degree Paper)

VILNIUS GEDIMINAS TECHNICAL UNIVERSITY

Jorge Diosdado Borrego, 20140135

(Student's given name, family name, certificate number)

Faculty of Electronics

(Faculty)

Automation, ATvf-10/1

(Study programme, academic group no.)

**DECLARATION OF AUTHORSHIP
IN THE FINAL DEGREE PAPER**

June 10, 2014

I declare that my Final Degree Paper entitled „Development of Adaptive Trajectory Generation Method for a Hexapod Robot“ is entirely my own work. The title was confirmed on June 6, 2014 by Faculty Dean's order No. 126el. I have clearly signalled the presence of quoted or paraphrased material and referenced all sources.

I have acknowledged appropriately any assistance I have received by the following professionals/advisers: Assoc Prof Dr Bronius Karaliūnas.

The academic supervisor of my Final Degree Paper is Dr Tomas Luneckas.

No contribution of any other person was obtained, nor did I buy my Final Degree Paper.



(Signature)

Jorge Diosdado Borrego

(Given name, family name)

INDEX

1. Introduction.....	7
1.2 Objectives.....	7
2. Analytical part.....	9
2.1 Stationary robots.....	9
2.2 Locomotion overview.....	12
2.3 Wheeled robots.....	15
2.4 Legged robots.....	17
2.5 Hexapod robots.....	20
3 Design part.....	23
3.1 Electronics schemes of the hexapod robot.....	23
3.2 Kinematics.....	25
3.3 Construction of trajectory generation function.....	28
3.4 Implementing the parabola in the MATLAB® model.....	33
3.5 Adaptive trajectories method.....	35
3.6 Obstacles.....	37
4 Conclusions.....	50
4.1 Future work.....	50
5 References	51

1. INTRODUCTION

As a first approach to understand this final thesis, it is important to understand what a robot is. One of the simplest and best definitions for a robot is the following one.

A robot is a mechanical or virtual artificial agent, usually an electro-mechanical machine that is guided by a computer program or electronic circuitry. Robots can be autonomous or semi-autonomous and range from humanoids to industrial robots, collectively programmed 'swarm' robots, and even microscopic nano-robots [10].

The branch of technology that deals with the design, construction, operation, and application of robots, as well as computer systems for their control is robotics. These technologies deal with automated machines that can take the place of humans in dangerous environments or manufacturing processes, or resemble humans in appearance, behavior, and/or cognition.

It is a fact, that robotics is a field of study that is developing very fast, and there are still a lot of problems or just tedious task that can easily be accomplished by robots. If we success in using robots in the right way, and having in mind our moral values, the society can advance in good way. This is one of the reasons why I chose to work in robotics in my final thesis for my bachelor.

This thesis explains the work I developed in this field, during the second semester of the school year 2013-2014.

Objectives

The main goal of this thesis is to develop a method for trajectory generation for a given hexapod robot.

The robot is shown in figure 1.1.

The ultimate goal of developing adaptive trajectories for the robot can be structured into two big stages, which can be clearly seen along the experiments.

1. Mathematics of trajectories

In this work, it has been carried out a thorough study of the best mathematics functions that the legs of the robot might follow, obtaining a parabola as the best mathematic function for this purpose.

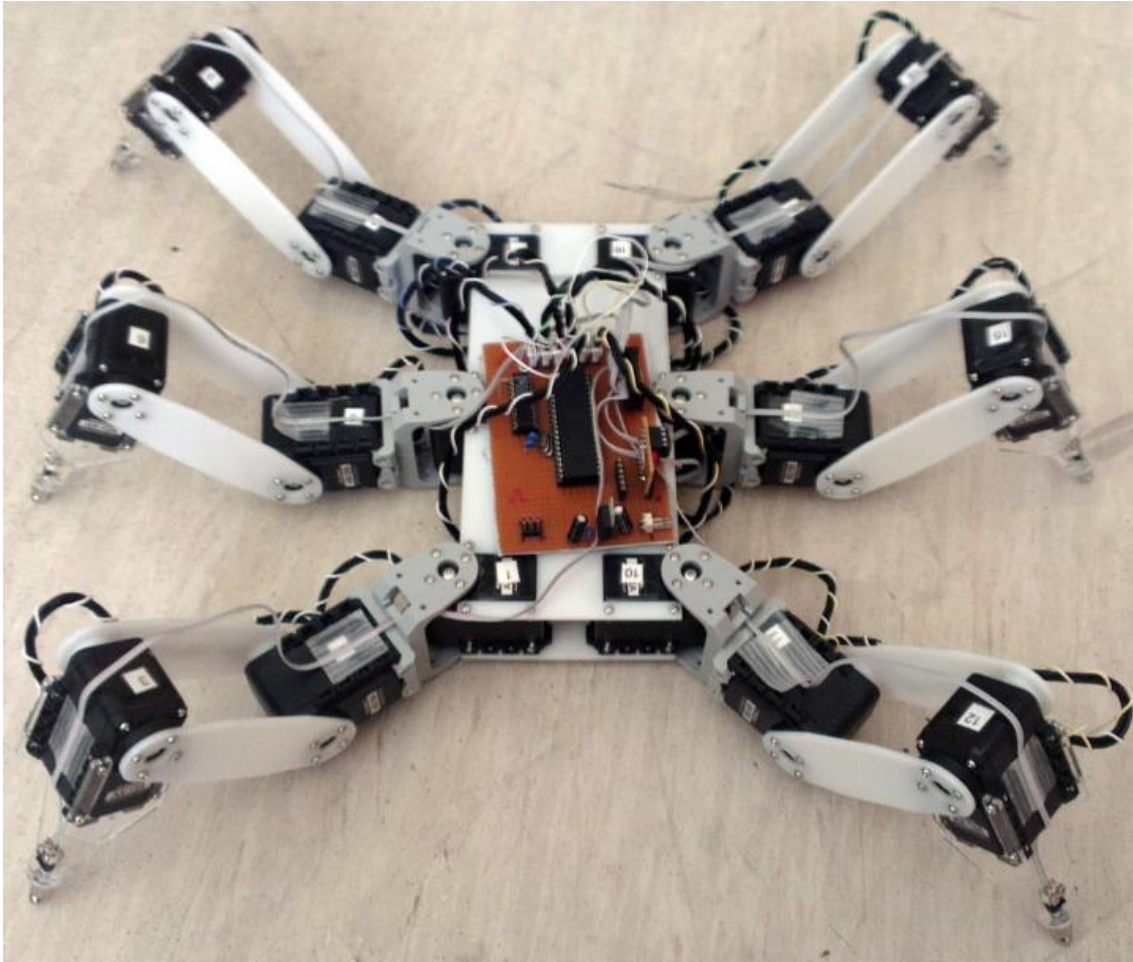


Figure 1.1 Hexapod robot

2. Check trajectory generation with MATLAB® code which simulates the robot

Before implementing any new technic or program in a machine, it is always easier to check it on a simulator first. Particularly with complex robots, it is nearly a necessity, since there are a lot of variables that can be checked and modified as desired towards a specific goal.

For that, the hexapod robot of study, have an accurate simulator constructed in MATLAB ®. This is the application where all the new ideas and movements for the robot were checked.

2. ANALYTICAL PART

There are several classifications of robots. Robots can be classified according to applications, number of degrees of freedom etc. Since this thesis' purpose is development of a method for trajectory generation, it appears logical to discuss about means of locomotion and classify robots with the same criteria.

2.1 Stationary robots

This category includes robotic industrial arms with a global axis of movement. Although the hexapod robot is not a stationary one, these robots comprise a very large and important group. That is the reason why they should be reviewed in a short way.

Cartesian/Gantry robots: A cartesian coordinate robot (also called linear robot) is an industrial robot whose three principal axes of control are linear (i.e. they move in a straight line rather than rotate) and are at right angles to each other. The three sliding joints correspond to moving the wrist up-down, in-out, back-forth. Among other advantages, this mechanical arrangement simplifies the robot control arm solution. Cartesian coordinate robots with the horizontal member supported at both ends are sometimes called Gantry robots [11].

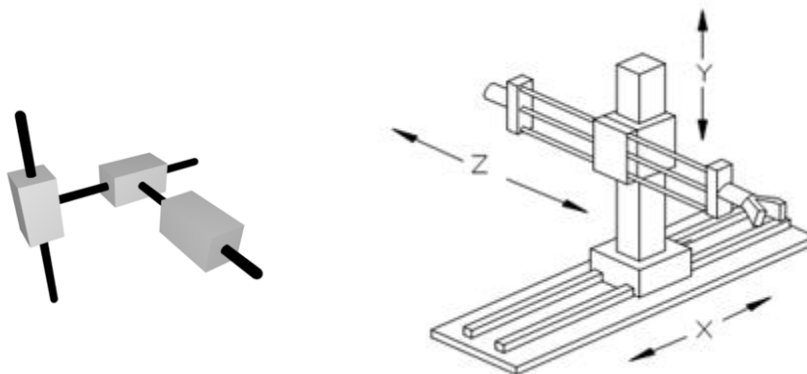


Figure 2.1. Cartesian Robot

Cylindrical robots: A cylindrical robot has two linear axes and one rotary axis. The robot derives its name from the operating envelope (the space in which a robot operates that is created by moving the axes from limit to limit).

The Z axis is located inside the base, resulting in a compact end-of-arm design that allows the robot to "reach" into tight work envelopes without sacrificing speed or repeatability [12].



Figure 2.2. Cylindrical robot.

Spherical robots: A spherical robot is a robot with two rotary joints and one prismatic joint; in other words, two rotary axes and one linear axis. Spherical robots have an arm which forms a spherical coordinate system [17].

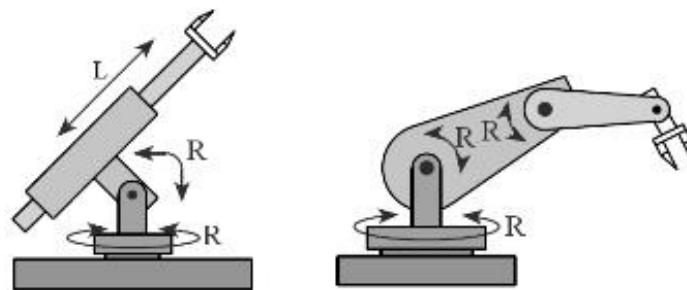


Figure 2.3. Cylindrical robot.

SCARA (Selective Compliant Assembly Robot Arm) robots: It is a type of cylindrical robot that has 4 axes of movement: X, Y, Z, and Theta Z. By virtue of the SCARA's parallel-axis joint layout, the arm is slightly compliant in the X-Y direction but rigid in the 'Z' direction, hence the term: Selective Compliant. This is advantageous for many types of assembly operations, i.e. inserting a round pin in a round hole without binding.

The second attribute of the SCARA is the jointed two-link arm layout similar to our human arms, hence the often-used term, articulated. This feature allows the arm to extend into confined areas and then retract or “fold up” out of the way. This is advantageous for transferring parts from one cell to another or for loading/unloading process stations that are enclosed.

SCARA's are generally faster and cleaner than comparable Cartesian robot systems. Their single pedestal mount requires a small footprint and provides an easy, unhindered form of mounting [7].

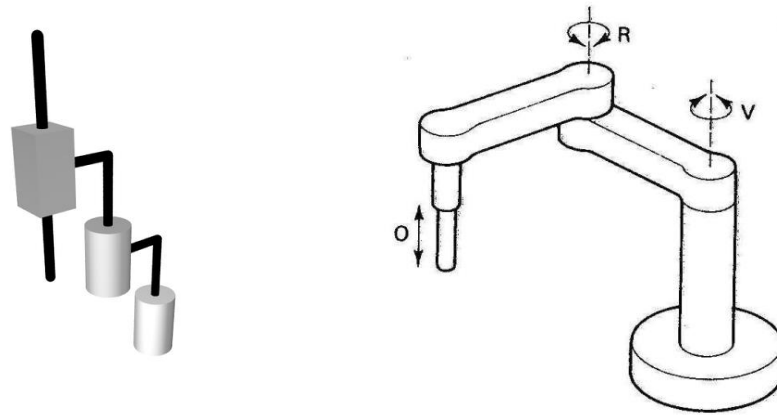


Figure 2.4. SCARA robot.

Articulated robots (robotic arms): An articulated robot is one which uses rotary joints to access its work space. Usually the joints are arranged in a “chain”, so that one joint supports another further in the chain.

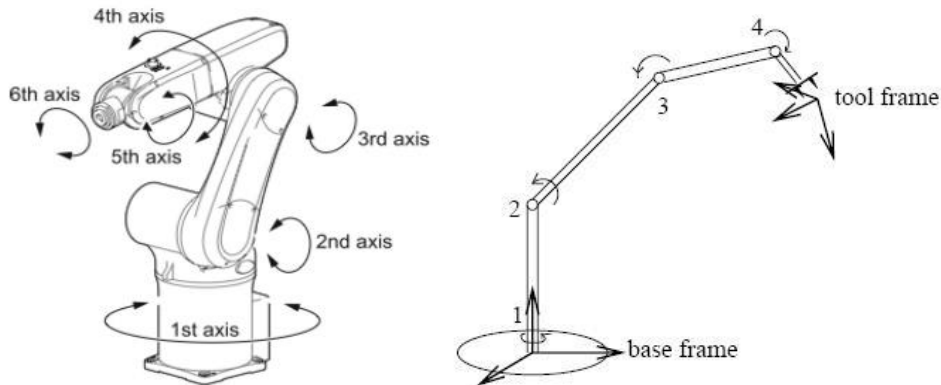


Figure 2.5. Articulated robot examples.

Parallel robots: parallel robots, or generalized Stewart platforms (in the Stewart platform, the actuators are paired together on both the basis and the platform), these systems are articulated robots that use similar mechanisms for the movement of either the robot on its base, or one or more manipulator arms. Their 'parallel' distinction, as opposed to a serial manipulator, is that the end effector (or 'hand') of this linkage (or 'arm') is connected to its base by a number of (usually three or six) separate and independent linkages working in parallel. 'Parallel' is used here in the computer science sense, rather than the geometrical; these linkages act together, but it is not

implied that they are aligned as parallel lines; here parallel means that the position of the end point of each linkage is independent of the position of the other linkages [13].

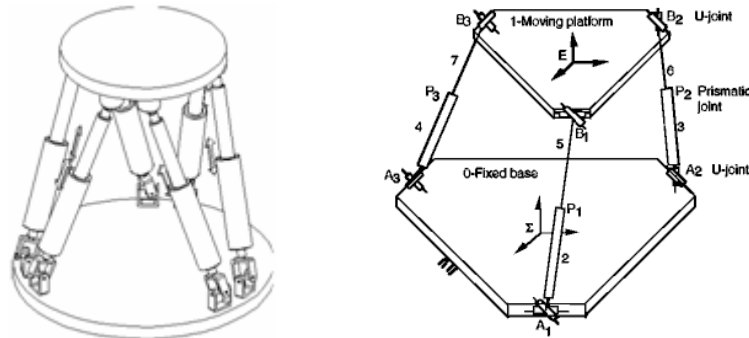


Figure 2.6. Parallel robot examples.

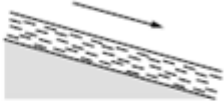


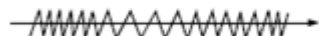

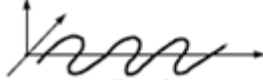






2.2 Locomotion overview

A mobile robot needs locomotion mechanisms that enable it to move as desired throughout its environment. But there are a large variety of possible ways to move, and so the selection of a robot's approach to locomotion is an important aspect of mobile robot design. In the laboratory, there are research robots that can walk, jump, run, slide, skate, swim, fly, and, of course, roll. Most of these locomotion mechanisms have been inspired by their biological counterparts (see table 1) [3].

There is, however, one exception: the actively powered wheel is a human invention that achieves extremely high efficiency on flat ground. This mechanism is not completely foreign to biological systems. Our bipedal walking system can be approximated by a rolling polygon, with sides equal in length to the span of the step (figure 2.7). As the step size decreases, the polygon approaches a circle or wheel. But nature did not develop a fully rotating, actively powered joint, which is the technology necessary for wheeled locomotion. Biological systems succeed in moving through a wide variety of harsh environments. Therefore it can be desirable to copy their selection of locomotion mechanisms. However, replicating nature in this regard is extremely difficult for several reasons. To begin with, mechanical complexity is easily achieved in biological systems through structural replication [3].

Cell division, in combination with specialization, can readily produce a millipede with several hundred legs and several tens of thousands of individually sensed cilia. In manmade structures, each part must be fabricated individually, and so no such economies of scale exist. Additionally, the cell is a microscopic building block that enables extreme miniaturization. With very small size and weight, insects achieve a level of robustness that we have not been able to match with human fabrication techniques. Finally, the biological energy storage system and the muscular and hydraulic activation systems used by large animals and insects achieve torque, response time, and conversion efficiencies that far exceed similarly scaled man-made systems [3].

Table 1. Locomotion mechanisms used in biological systems.

Type of motion	Resistance to motion	Basic kinematics of motion
Flow in a Channel 	Hydrodynamic forces	Eddies 
Crawl 	Friction forces	Longitudinal vibration 
Sliding 	Friction forces	Transverse vibration 
Running 	Loss of kinetic energy	Oscillatory movement of a multi-link pendulum 
Jumping 	Loss of kinetic energy	Oscillatory movement of a multi-link pendulum 
Walking 	Gravitational forces	Rolling of a polygon 

Owing to these limitations, mobile robots generally locomote either using wheeled mechanisms, a well-known human technology for vehicles, or using a small number of articulated legs, the simplest of the biological approaches to locomotion (see figure 2.7). In general, legged locomotion requires higher degrees of freedom and therefore greater mechanical complexity than wheeled locomotion. Wheels, in addition to being simple, are extremely well suited to flat ground. As figure 2.8 depicts, on flat surfaces, wheeled locomotion is one to two orders of magnitude more efficient than legged locomotion. But as the surface becomes soft, wheeled locomotion accumulates inefficiencies due to rolling friction whereas legged locomotion suffers much less because it consists only of point contacts with the ground. This is demonstrated in figure 2.8 by the dramatic loss of efficiency in the case of a tire on soft ground [3].

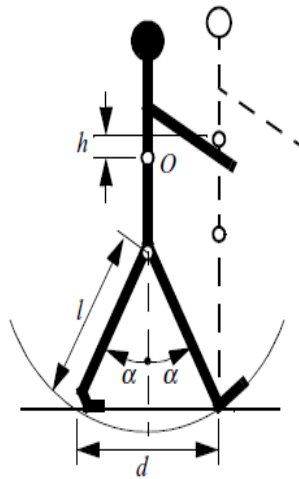


Figure 2.7. A biped walking system can be approximated by a rolling polygon, with sides equal in length d to the span of the step. As the step size decreases, the polygon approaches a circle or wheel with the radius l [3].

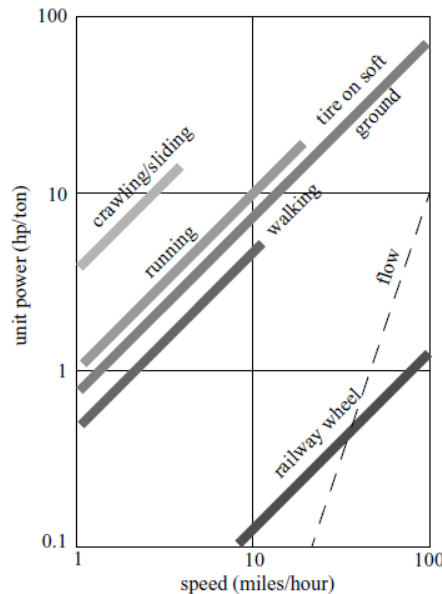


Figure 2.8. Specific power versus attainable speed of various locomotion mechanisms [3].

In effect, the efficiency of wheeled locomotion depends greatly on environmental qualities, particularly the flatness and hardness of the ground, while the efficiency of legged locomotion depends on the leg mass and body mass, both of which the robot must support at various points in a legged gait.

It is understandable therefore that nature favors legged locomotion, since locomotion systems in nature must operate on rough and unstructured terrain. For example, in the case of insects in a forest the vertical variation in ground height is often an order of magnitude greater than the total height of the insect. By the same token, the human environment frequently consists of engineered, smooth surfaces, both indoors and outdoors. Therefore, it is also understandable that virtually all industrial applications of

mobile robotics utilize some form of wheeled locomotion. Recently, for more natural outdoor environments, there has been some progress toward hybrid and legged industrial robots [3].

2.3 Wheeled robots

Wheeled robots are robots that navigate around the ground using motorized wheels to propel themselves. This design is simpler than using treads or legs and by using wheels they are easier to design, build, and program for movement in flat, not-so-rugged terrain. They are also more well controlled than other types of robots. However, some disadvantages of wheeled robots are that they cannot navigate well over obstacles, such as rocky terrain, sharp declines, or areas with low friction. [18]

Wheeled robots are most popular among the consumer market because their differential steering provides low cost and simplicity. Robots can have any number of wheels, but three wheels are sufficient for static and dynamic balance.

Two wheeled robots: Two wheeled robots are harder to balance than other types because they must keep moving to maintain upright. The center of gravity of the robot body is kept below the axle, usually this is accomplished by mounting the batteries below the body.

Famous examples of two wheeled robots are Roomba and Segway transporter.

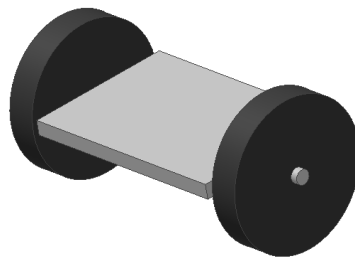


Figure 2.9. Two wheeled robot.

Three wheeled robots: 3-wheeled vehicles may be of two types: differentially steered (2 powered wheels with an additional free rotating wheel to keep the body in balance) or 2 wheels powered by a single source and a powered steering for the third wheel. In the case of differentially steered wheels, the robot direction may be changed by varying the relative rate of rotation of the two separately driven wheels. If both the wheels are driven in the same direction and speed, the robot will go straight. Otherwise, depending on the speed of rotation and its direction, the center of rotation may fall anywhere in the line joining the two wheels [14].

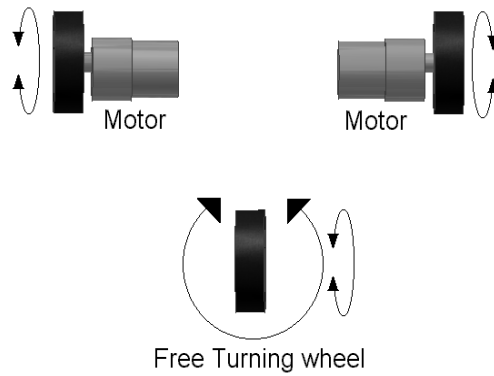


Figure 2.10. Differentially steered 3 wheeled robot.

Four wheeled robots: four wheeled robots may be of two types:

- 2 powered, 2 free rotating wheels: the robot direction may be changed by varying the relative rate of rotation of the two separately driven wheels [19].

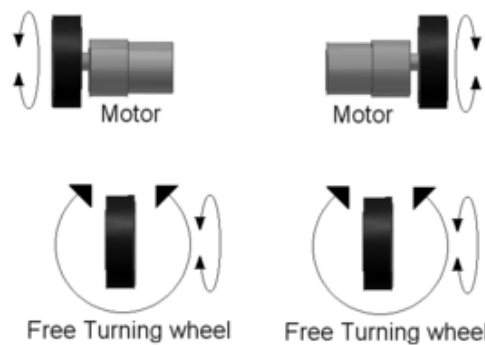


Figure 2.11. wheels powered, 2 free rotating wheels.

- 2-by-2 powered wheels for tank-like movement: This kind of robot uses 2 pairs of powered wheels. Each pair (connected by a line) turn in the same direction. The tricky part of this kind of propulsion is getting all the wheels to turn with the same speed. If the wheels in a pair aren't running with the same speed, the slower one will slip (inefficient). If the pairs don't run at the same speed the robot won't be able to drive straight. A good design will have to incorporate some form of car-like steering. [14].

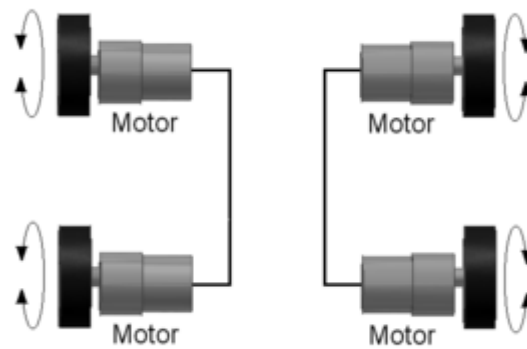


Figure 2.12. 2-by-2 powered wheels for tank-like movement.

– Car-like steering: This method allows the robot to turn in the same way a car does.. This system does have an advantage over previous methods when your robot is powered by a combustion engine: It only needs one motor (and a servo for steering of course). The previous methods would require either 2 motors or a very complicated gearbox, since they require 2 output axes with independent speed and direction of rotation [8].

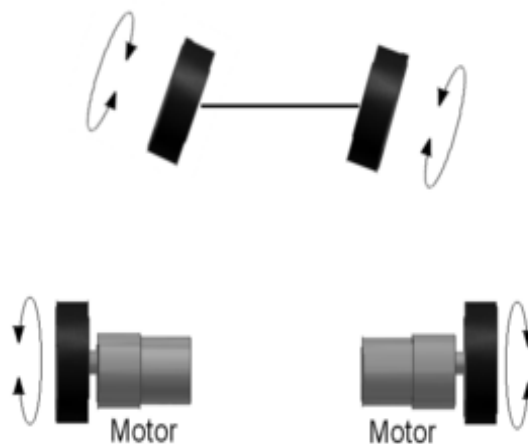


Figure 2.13. Car-like steering.

2.4 Legged robots

Legged locomotion is characterized by a series of point contacts between the robot and the ground. The key advantages include adaptability and maneuverability in rough terrain because only a set of point contacts is required; the quality of the ground between those points does not matter so long as the robot can maintain adequate ground clearance. In addition, a walking robot is capable of crossing a hole or chasm as long as its reach exceeds the width of the hole. Another advantage of legged locomotion is the

potential to manipulate objects in the environment with great skill. An excellent insect example, the dung beetle, is capable of rolling a ball while locomoting by way of its dexterous front legs. And as a final advantage, legs create discrete footprints and cause less damage to natural terrain.

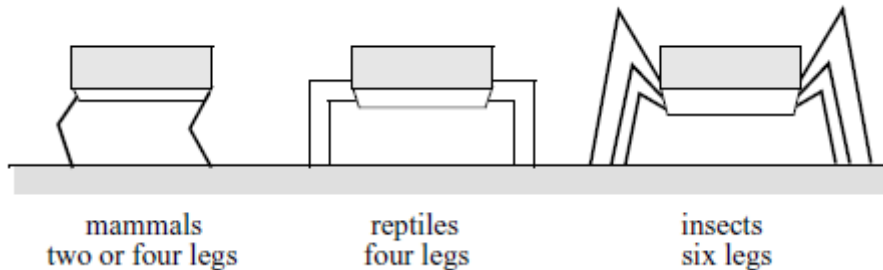


Figure 2.14. Car-like steering [3].

The main disadvantages of legged locomotion include power and mechanical complexity. The leg, which may include several degrees of freedom, must be capable of sustaining part of the robot's total weight, and in many robots must be capable of lifting and lowering the robot. Additionally, high maneuverability will only be achieved if the legs have a sufficient number of degrees of freedom to impart forces in a number of different directions [3].

With the advances in control of complex systems, efforts to develop legged machines have become more intense.

Leg configuration and stability

Since all legged robots are inspired by nature, it is a good approach to examine biologically successful legged systems, in order to gain knowledge and experience. As shown in figure 2.14, different configurations for legs have been successful in different animals. Large animals, such as mammals and reptiles, have four legs, whereas insects have six or more legs. In some mammals, the ability to walk on only two legs has been perfected. Especially in the case of humans, balance has progressed to the point that we can even jump with one leg. This exceptional maneuverability comes at a price: much more complex active control to maintain balance [3].

In contrast, a creature with three legs can exhibit a static, stable pose provided that it can ensure that its center of gravity is within the tripod of ground contact. Static stability, demonstrated by a three-legged stool, means that balance is maintained with no need for motion. A small deviation from stability (e.g., gently pushing the stool) is passively corrected toward the stable pose when the upsetting force stops [3].

But a robot must be able to lift its leg at the end of its effective stroke, return it, and place it on the ground to begin another support phase. In order to achieve static walking, a robot must have at least six legs generally. In such a configuration, it is possible to design a gait in which a statically stable tripod of legs is in contact with the ground at all times [3].

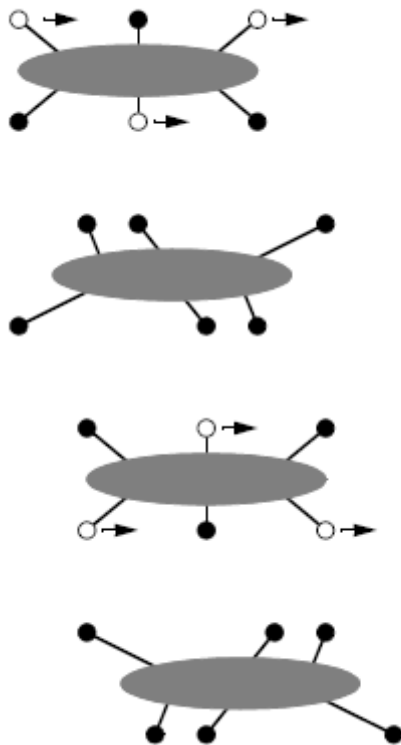


Figure 2.15. Static walking with six legs [3]

In the case of legged mobile robots, a minimum of two degrees of freedom is generally required to move a leg forward by lifting the leg and swinging it forward. More common is the addition of a third degree of freedom for more complex maneuvers, resulting in legs such as those shown in figure 2.14. Recent successes in the creation of bipedal walking robots have added a fourth degree of freedom at the ankle joint. The ankle enables more consistent ground contact by actuating the pose of the sole of the foot.

In general, adding degrees of freedom to a robot leg increases the maneuverability of the robot, both augmenting the range of terrains on which it can travel and the ability of the robot to travel with a variety of gaits. The primary disadvantages of additional joints and actuators are, of course, energy, control, and mass. Additional actuators require energy and control, and they also add to leg mass, further increasing power and load requirements on existing actuators.

In the case of a multi-legged mobile robot, there is the issue of leg coordination for locomotion, or gait control. The number of possible gaits depends on the number of legs [3].

The gait is a sequence of lift and release events for the individual legs. For a mobile robot with k legs, the total number of possible events N for a walking machine is $N = (2k - 1)!$

For a biped walker ($k=2$) legs, the number of possible events N is

$$N = (2k - 1)! = 3! = 3 \cdot 2 \cdot 1 = 6.$$

The six different events are:

1. lift right leg;
2. lift left leg;
3. release right leg;
4. release left leg;
5. lift both legs together;
6. release both legs together.

Of course, this quickly grows quite large. For example, a robot with six legs, like the robot studied here, has far more gaits theoretically:

$$N = 11! = 39916800 [3].$$

2.5 Hexapod robots

As said before, hexapod configuration for robots, is the best for obtaining static walking and therefore, this leads to a less complex control of the robots. To discuss hexapod gaits and movement, the study of insects become very fruitful, as they are arguably the most successful locomoting creatures on earth, excel at traversing all forms of terrain with six legs, even upside down.

The following figures show some examples of hexapod robots.



Figure 2.16. Examples of different hexapod robots.

First, there are some definitions that need to be explained, in order to understand the section:

Protraction is the forward movement of a leg relative to the body and ground. *Retraction* is the backward movement of a leg relative to the body with no movement of the leg relative to the ground.

The *transfer phase* (or swing) of a leg is the period in which the leg is not on the ground.

The *support* (or stance) of a leg is the period in which the leg is on the ground.

The *cycle time* is the time for a complete cycle of locomotion.

The *duty factor* of a leg is the time fraction of cycle time for which that leg is in the support phase.

The *leg phase* of a leg is the fraction of cycle time by which the contact of that leg on the ground lags behind the contact of the front leg of left side.

The *stride length* is the distance the center of gravity translates during one complete locomotion cycle [4].

The limb movements of many arthropods occur in metachronal sequences (each leg lifts when the leg behind it is on ground), often sequences running from posterior to anterior. Wilson showed that many of the common gaits observed in insects could be generated by changes in just one variable namely the time. In particular, Wilson [2] accounted for the smooth transition between a metachronal gait at low walking speeds and an alternating tripod gait at high speeds. As per Wilson hypothesis, leg movement in insects follows some general rules. These are:

1. Forward movement of legs relative to the body runs from posterior to anterior and no leg protracts until the one behind is placed in a supporting position.
2. Contra lateral legs of the same body segment alternate in phase.
3. Protraction time is constant.
4. Retraction time decreases as frequency of stepping increases.
5. The intervals between steps of the hind leg and fore leg and between the middle leg and fore leg are constant. However, the interval between the fore leg and the hind leg steps varies inversely with stepping frequency [4].

According to Wilson, no other patterns, using six legs have been reported for straight walking. Later, he added modifying conditions to accommodate much of the experimental evidences from other animals. However, in special cases, the model is either inaccurate or does not hold at all. For example, climbing grasshopper moves the two legs of a segment together. At slow speeds, mantis is also functionally quadrupedal using only the posterior two pairs of legs. Some other scientist made a detailed study on the cockroach *Periplaneta americana* and found nearly linear relationship between average frequency of leg movement and rate of forward progression [4].

The stride length of the insect is more or less constant. Except at very low speeds (stepping frequency less than 3 Hz), it always uses the alternating tripod gait. At higher speeds, it runs with its body raised well off the ground and its anterior end elevated relative to the posterior.

It switches to a gait using only four hind legs. In this posture, the body is propelled primarily by the long hind legs. Increase in angle of attack of the body also allows greater range of movement through which the hind legs can swing [4].

Another advantage of hexapod robots is that they can walk forward with many kinds of gaits (see figure 2.17) to adapt different speeds and loads. And because the redundant limb exists, hexapod robot could continue its work even if limb is lost.

These advantages makes it competent for some autonomous and high-reliability works, such as field scouting, underwater searching, and space exploring. However, the control of those 6 legs becomes a complex task, due to the number of variables that must be

monitored. There is large body of work devoted to the control of hexapod robot which contains gait planning and kinematics control.

Gait takes an important role in the control of walking machine. The gait synthesis might be based on kinematics model of the robot and walking rules that are well known from insect walking. There are two main types of gaits adopted in walking machines: periodic gait and non-periodic gait which is also called free gait.

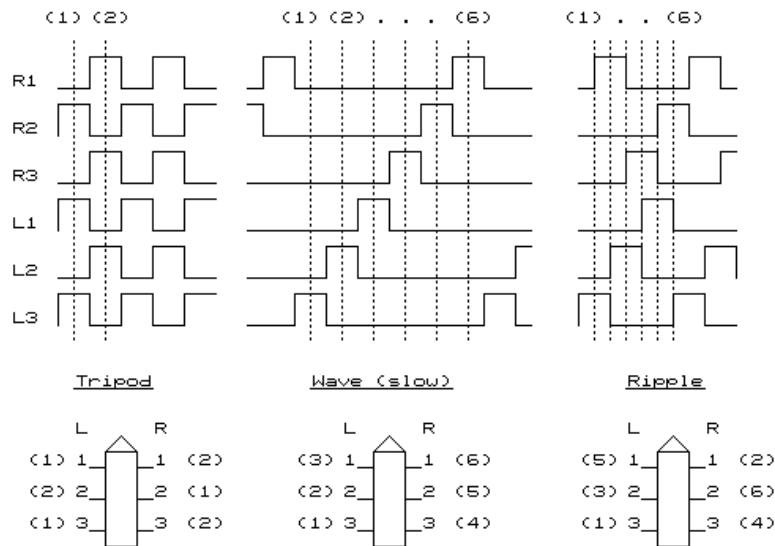


Figure 2.17. Tripod, wave and ripple gaits for a hexapod robot.

The free gait increases the adaptability of the walking machine because it can move on uneven terrain. However, the free gait is hard to be realized in the real multi-legged walking robots and is only on the stage of theoretical research. Periodic gait such as tripod gait can be easily controlled and has an optimal stability margin.

Since tripod gait seems to be the most appropriate for hexapod robot, it is the one chosen to start programming and testing.

3. DESIGN PART

3.1 Electronics schemes of the hexapod robot

In this section, the explanation of the electric scheme is showed. The main circuit is shown in figure 3.1.

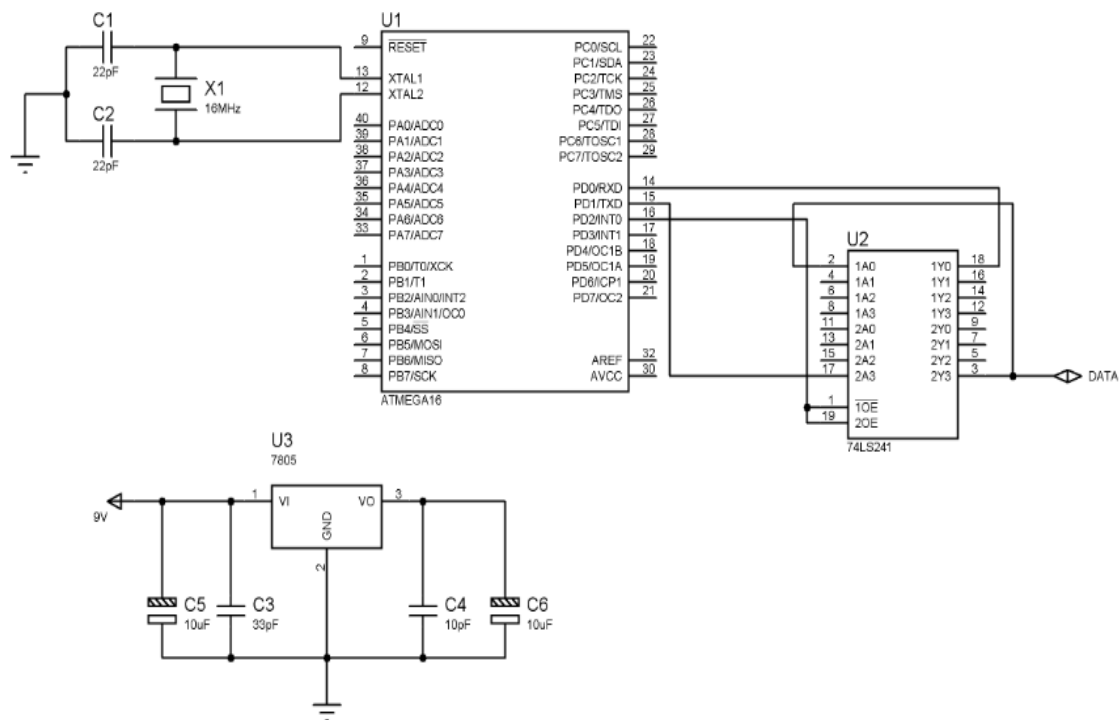


Figure 3.1. Electric robot's scheme.

U1 circuit

U1 circuit represents the microcontroller. It is an ATmega16. Some of the most important features are the following:

The ATmega16 provides the following features: 16 Kbytes of In-System Programmable Flash Program memory with Read-While-Write capabilities, 512 bytes EEPROM, 1 Kbyte SRAM, 32 general purpose I/O lines, 32 general purpose working registers, a JTAG interface for Boundary-scan, On-chip Debugging support and programming, three flexible Timer/Counters with compare modes, Internal and External Interrupts, a serial programmable USART and a byte oriented Two-wire Serial Interface [9].

In the circuit, the USART is used to receive and transmit signals to the U2 circuit.

- PD0 is the USART input pin
- PD1 is the USART output pin

In the microcontroller, it is where all the calculations are carried out. The two main functions developed in the microcontroller are the control of servos and the calculations of kinematics.

U2 circuit

This circuit is a Motorola SN54/74LS240. It transforms from two different reception and transmission lines to only one line. This is needed because the main controller communicates with the servos by sending and receiving data packets. There are two types of packets; the “Instruction Packet” (sent from the main controller to the Dynamixel actuators) and the “Status Packet” (sent from the actuators to the main controller). Since the controller needs two different lines for inputs and outputs, and the servo uses only one bi-directional data line, there is a need to convert from two lines in the microcontroller to one line in the servos, and this purpose is served by U2 circuit.

Some of the most important features are:

- Hysteresis at Inputs to Improve Noise Margins.
- 3-State Outputs Drive Bus Lines or Buffer Memory Address Registers.
- Input Clamp Diodes Limit High-Speed Termination Effects.

U3 circuit

This circuit is a power source, which transforms 12V input to 5V in the output. It is needed because ATmega16 is powered with 5V, whereas servos are powered with 12 V. With this circuit, we can use only one power source instead of two different ones, and this is very beneficial, as there is no need to carry two power sources, which would increase the weight.

Servos

The actuators used for the hexapod are Dynamixel AX-12 servos.



Figure 3.2. Dynamixel AX-12 servomotors

A total number of 18 are necessary to obtain 3 degrees of freedom in each leg. The main characteristics of the servos are showed below.

Table 2. Servos characteristics

	AX-12	
Weight (g)	55	
Gear Reduction Ratio	1/254	
Input Voltage (V)	at 7V	at 10V
Final Max Holding Torque(kgf.cm)	12	16.5
Sec/60degree	0.269	0.196

Resolution	0.35°
Operating Angle	300°, Endless Turn
Voltage	7V~10V (Recommended voltage: 9.6V)
Max. Current	900mA
Operate Temperature	-5 °C ~ +85 °C
Command Signal	Digital Packet
Protocol Type	Half duplex Asynchronous Serial Communication (8bit, 1stop, No Parity).
Link (Physical)	TTL Level Multi Drop (daisy chain type Connector)
ID	254 ID (0~253)
Communication Speed	7343bps ~ 1 Mbps
Feedback	Position, Temperature, Load, Input Voltage, etc.
Material	Engineering Plastic

3.2 Hexapod robot kinematics

Forward kinematics

Forward kinematics refers to the use of the kinematic equations of a robot to compute the position of the end-effector from specified values for the joint parameters [5]. In this case, the end effector is considered to be the end of the robot's leg.

The most common and easy way to obtain forward kinematics is Denavit-Hartenberg algorithm, and this is the method used in this work.

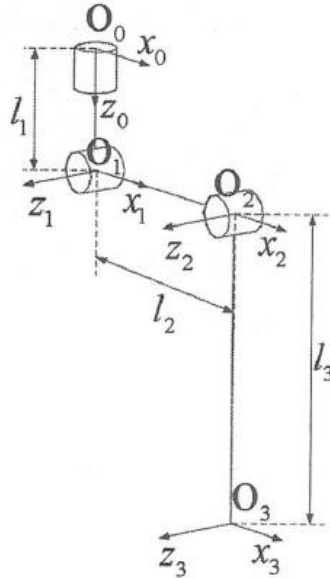


Figure 3.3. Kinematic layout for one leg.

From leg kinematic layout showed in figure 3.3, Denavit-Hartenberg solution gives the following three equations:

$$\begin{aligned}
 x &= \cos\theta_1 \cos\theta_2 l_3 \cos\theta_3 - \cos\theta_1 \sin\theta_2 l_3 \sin\theta_3 + \cos\theta_1 l_2 \cos\theta_2 \\
 y &= \sin\theta_1 \cos\theta_2 l_3 \cos\theta_3 - \sin\theta_1 \sin\theta_2 l_3 \sin\theta_3 + \sin\theta_1 l_2 \cos\theta_2 \quad (1) \\
 z &= -\sin\theta_2 l_3 \cos\theta_3 - \cos\theta_2 l_3 \sin\theta_3 - l_2 \sin\theta_2 - l_1
 \end{aligned}$$

These equations provide a relation between the position of robot's foot and the angle of the servos used as actuators. As it can be observed, they provide the foot position when these angles are known, which means forward kinematics.

Inverse Kinematics

In order to get inverse kinematics expressions, it is necessary to express θ_1 , θ_2 and θ_3 over x , y and z . Such task could be very complex or even unsolvable.

Less complex way of dealing with robot's inverse kinematics is geometric inverse kinematics method. The following figures demonstrate kinematic layout of one leg for geometric inverse kinematics (Figure 3.4) [1].

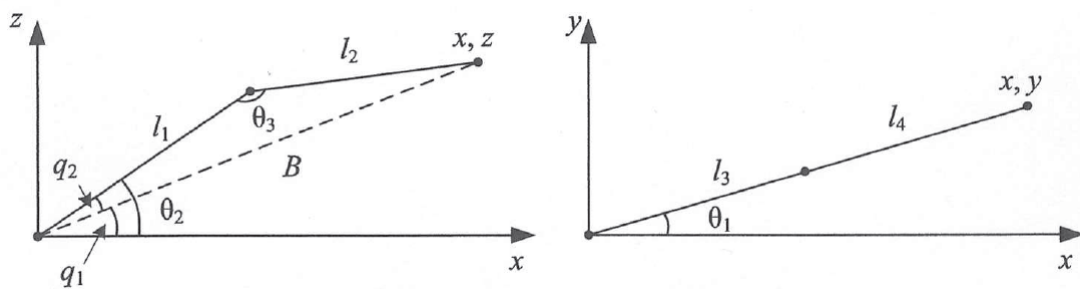


Figure 3.4. Leg projection onto XZ and XY planes.

The following expressions are derived using this method:

$$\begin{aligned}
\theta_1 &= \arctan \frac{y}{x+l_3}, \\
\theta_2 &= q_1 + q_2, \\
\theta_3 &= \arccos \frac{l_1^2 + l_2^2 - B}{2l_1 l_2}, \\
l_4 &= \sqrt{x^2 + y^2} + l_3, \\
B &= \sqrt{(l_4 - l_3)^2 + z^2}, \\
q_1 &= \arcsin \frac{z}{B}, \\
q_2 &= \arccos \frac{l_1^2 - l_2^2 + B}{2l_1 B}.
\end{aligned} \tag{2}$$

Where θ_1 , θ_2 and θ_3 are leg actuator angles that must be calculated in order to position robot's foot into position with coordinates x , y and z .

Transforming leg base coordinates frames into body coordinate frame

Once the kinematics for legs is solved, there is a need to transform each coordinate frame into robot's coordinate frame. This way, it is easy to correct body positions, for instance when robot is climbing on inclined surface. This makes possible also to lift robot's body when needed. Figure 3.5 shows the coordinate frames layout on the robot.

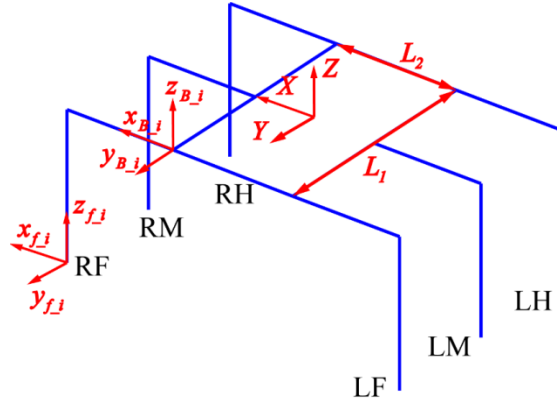


Figure 3.5. Coordinates frames on the robot.

To calculate leg base coordinate frame's coordinates in robot's body coordinate frame, homogeneous transformation matrices are used. Each leg's base coordinate frame is transformed using three rotations and one translation. The homogeneous transformation matrix (B_i) is obtained by multiplying the matrix of translation (T_i) by the matrix of rotation (T_{XYZ}). In this matrix, r_{ii} represents the rotation of the new base coordinates frame, and X_0, Y_0, Z_0 the translation of this new base coordinates.

$$B_i = T_{XYZ} \cdot T_i$$

$$T_{XYZ} = \begin{bmatrix} \mathbf{R}_{XYZ} & \mathbf{d}_{XYZ} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{11} & r_{11} & X_0 \\ r_{11} & r_{11} & r_{11} & Y_0 \\ r_{11} & r_{11} & r_{11} & Z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$B_i = \begin{bmatrix} X_{B_i} \\ Y_{B_i} \\ Z_{B_i} \\ 1 \end{bmatrix}, T_i = \begin{bmatrix} x_{B_i} \\ y_{B_i} \\ z_{B_i} \\ 1 \end{bmatrix}, \mathbf{d}_{XYZ} = \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix}, \quad (3)$$

3.3 Construction of trajectory generation function

In this section, the whole process followed to obtain the final method for trajectory generation is explained.

The robot was given performing sinusoidal trajectories for its legs. However, this seemed not the best trajectory function for a robot that should adapt to terrain. This is better explained with an example. When robot would encounter a hole, it would change concavity, going from a convex function to a concave function, and allowing situations as the one in figure 3.6 to happen. In this kind of situations, the leg would step on the “wall” of the hole, and it would unbalance the whole robot. To avoid this situation, another function was needed, with a more appropriate shape for the step.

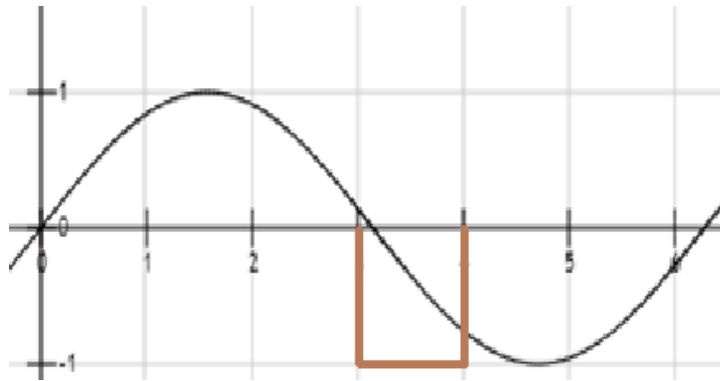


Figure 3.6. Sinusoidal function. Leg drops in a hole.

Parabola is one of the solutions that first come to mind. When compared with sinusoidal function (figure 3.7), it appears to be more suitable for adaptive trajectories.

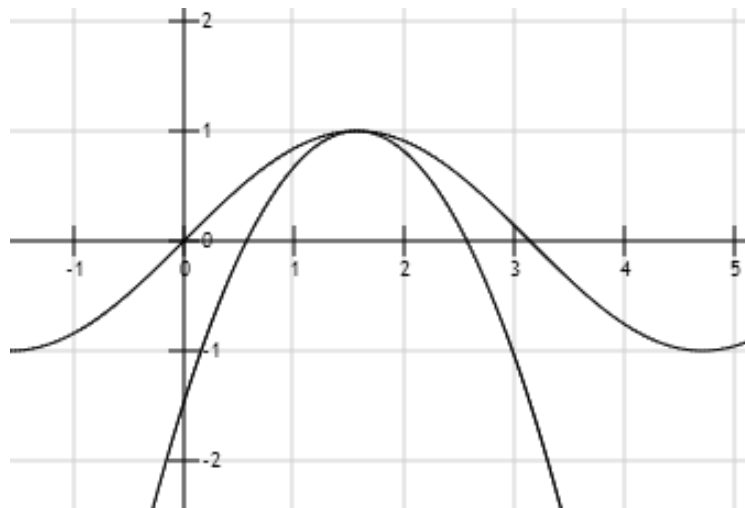


Figure 3.7. Sinusoidal and parabola functions.

The second advantage of this function is that, with some mathematical development, we can obtain a parabola from two important parameters: *step-length* and *step-height*. The idea is to input a step height suitable for the kind of obstacles robot can encounter, and a proper step length for them.

To achieve this, the work was divided in some steps that are explained below.

First approach

The first problem to solve is to build a function that implements a parabola providing two inputs: *step-height* and *step-length*.

```

1  function [] = parabola3 (a,b,c)
2
3  x=-10:0.01:10;
4
5
6  y=-a*x.^2+b*x+c;
7
8  plot (x,y);
9  grid on;
10 |

```

Figure 3.8. Parabola function.

This function plots a parabola; however, this mathematic expression for the parabola doesn't seem to be the most suitable for our purpose, because we are seeking an

expression that permits to plot a parabola from two geometrical parameters. This leads to another equation of parabola.

Second approach

In this second approach, the function implements a parabola, receiving 4 inputs: k , $stepheight$, p , a .

$Step-height$: height of the parabola;

k : longitudinal displacement;

p : changes the position of focus;

a : this parameter controls the shape of the parabola.

```
function [] = parabola2 (k, stepheight, p,a)
%k= longitudinal displacement in y (it will be 0)
%4p= straight segment of parabola
% focus is on (h,k+p)

% increasing a, parabola gets thinner:

x=-10:0.01:10;

y=-((a*x-k).^2)/(4*p)+stepheight;

plot (x,y);
hold on

plot(k,stepheight-p,'s');
y=stepheight-p;
plot(x,y,'g')

grid on;
```

Figure 3.9. New parabola function.

If we execute this function, the following shape is obtained.

```
>> parabola2_antigua(0,4,1,4)
```

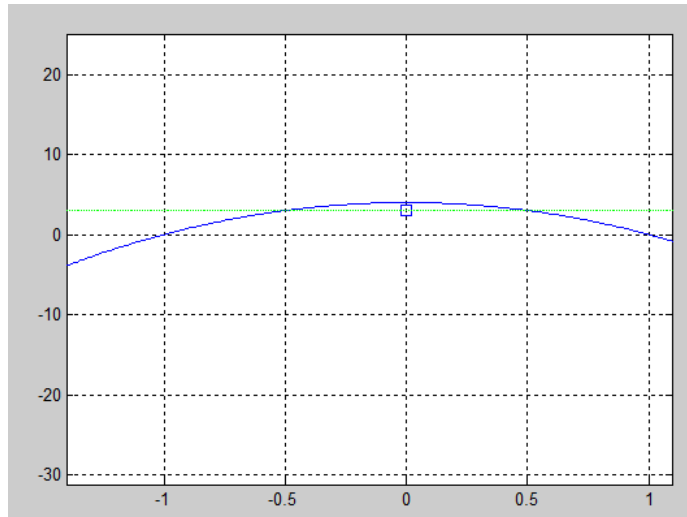


Figure 3.10. New parabola function shape.

The function works properly, but these parameters are not the desired ones. We are looking for a parabola with only three inputs (*longitudinal displacement, step-height, and step-length*). But this mathematic expression of the parabola, allows to find the desired formula for the parabola. We just need to do some mathematical operations.

If we take the equation of the parabola

$$Y = -\frac{(a*x-k)^2}{4*p} + k. \quad (4)$$

The *step-length* should be measured in $Y=0$, so:

$$0 = -\frac{(a*x-k)^2}{4*p} + \text{stepheight}, \quad (5)$$

$$\text{stepheight} * 4p = (ax)^2, \quad (6)$$

$$x = \frac{\sqrt{h*4p}}{a}. \quad (7)$$

And x is half the length of the step length. So step length is:

$$2x = 2 \frac{\sqrt{\text{stepheight}*4p}}{a} = \text{Step length}. \quad (8)$$

As we can see in (8), *step-length* depends on *step-height* and parameter a . It became clear then that parameter a cannot be an input of the function, because it is something already fixed and depending on *step-height*.

This leads to a new function, in which parameter a is calculated from *step-length* and *step-height*, and it is not an input of the function. This way, we can obtain the desired parabola, controlling step height and step length.

Obtaining parameter a is easy, and it comes from clearing it in the previous equation:

$$2 \frac{\sqrt{\text{stepheight} * 4p}}{a} = \text{Step length}, \quad (9)$$

$$2 \frac{\sqrt{\text{stepheight} * 4p}}{\text{Step length}} = a. \quad (10)$$

Parameter p is redundant now, because the *step-height* and *step-length* are enough to define one parabola. This is the new function:

```
function [] = parabola2 (k, stepheight, steplength)

%k= longitudinal displacement in y (it will be 0)
%4p= straight segment of parabola
% focus is on (h,k+p)

x=-10:0.01:10;

a=2*sqrt(stepheight*4)/steplength;

y=-((a*x-k).^2)/(4)+stepheight;

plot (x,y);

grid on;
```

Figure 3.11. Third version of the function parabola.

This is the plot obtained when executed:

```
>> parabola2(0,100,2)
```

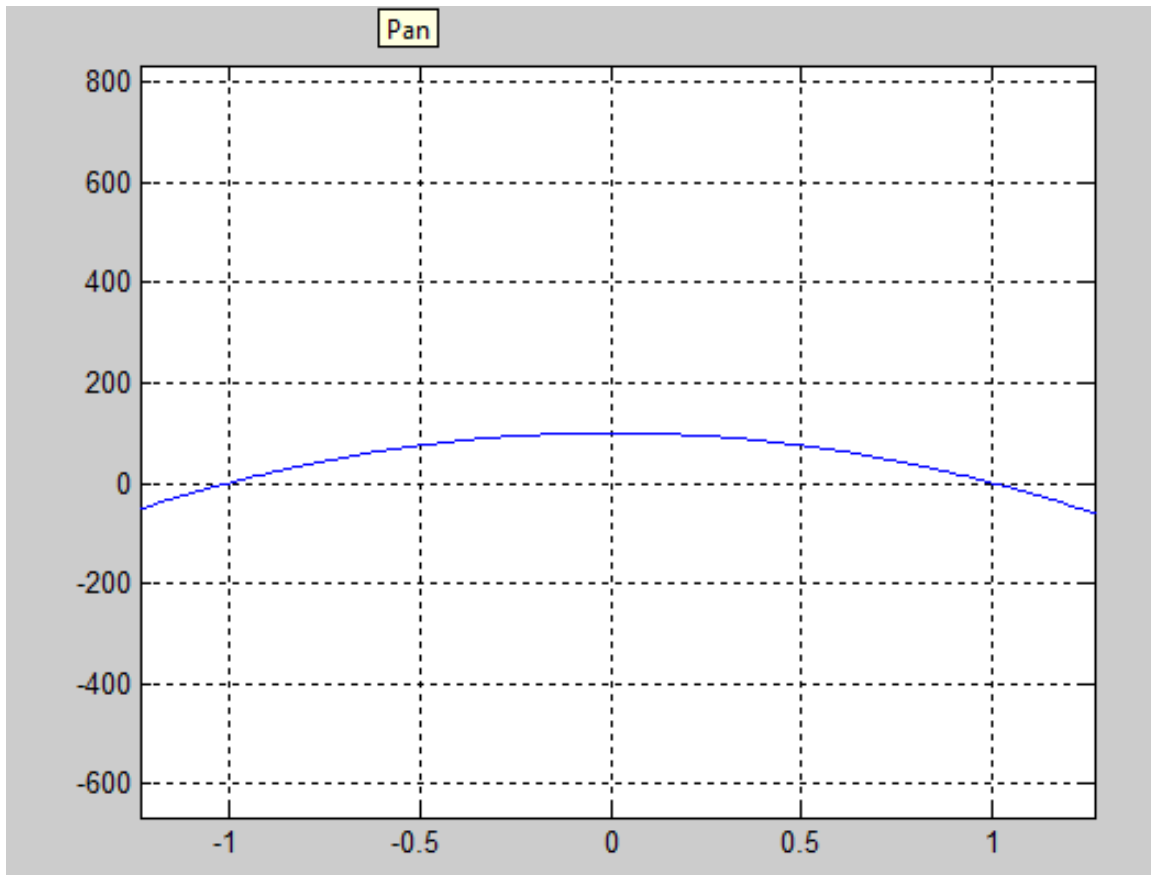



Figure 3.12. Parabola plot.

As it can be seen, now we obtain a parabola, with only the three desired inputs:

Step-length, Step-height and k (longitudinal displacement).

3.4 Implementing the parabola in the MATLAB® model of the robot

When mathematics are solved, it comes the time to implement the parabola in the robot model.

The function that needs to be changed is the following:

```

function [x, y, z] = feet_traj(phase, T, l, h, t, dir)
%if current time is less or equal than legs phase
if(t<=phase)
    x=-(l*sin(dir)*(T - 2*phase + 2*t - 1))/(2*(T - 1));
    y=-(l*cos(dir)*(T - 2*phase + 2*t - 1))/(2*(T - 1));
    z=0;
end
%if current time is more than legs phase and less that phase+1
if(t>phase)&&(t<=phase+1)
    x=(l*(t-phase-1/2))*sin(dir);
    y=(l*(t-phase-1/2))*cos(dir);
    z=h*sin((t-phase)*pi);
end
%if current time is more than legs phase+1
if(t>phase+1)
    x=(l*sin(dir)*(T + 2*phase - 2*t + 1))/(2*(T - 1));
    y=(l*cos(dir)*(T + 2*phase - 2*t + 1))/(2*(T - 1));
    z=0;
end

```

Figure 3.13. Feet_trajectory function.

The way to change the trajectory of feet is to change the coordinate z in the third case of the “if” loop [(t>phase)&&(t<=phase+1)].

Taking into account the way parabola is calculated, the parameter a , needs to be calculated before any case of the *if*.

The function remains then as shown in figure 3.14.

```

function [x, y, z] = feet_traj(phase, T, l, h, t, dir)
%if current time is less or equal than legs phase

a=2*sqrt(h*4)/l;

if(t<=phase)
    x=-(1*sin(dir)*(T - 2*phase + 2*t - 1))/(2*(T - 1));
    y=-(1*cos(dir)*(T - 2*phase + 2*t - 1))/(2*(T - 1));
    z=0;
end
%if current time is more than legs phase and less that phase+1
if(t>phase)&&(t<=phase+1)
    x=(1*(t-phase-1/2))*sin(dir);
    y=(1*(t-phase-1/2))*cos(dir);
    z=-((a*y).^2)/(4)+1;

    % cuando t=phase-> z=0; cuando t=phase+1->z=0
end
%if current time is more than legs phase+1
if(t>phase+1)
    x=(1*sin(dir)*(T + 2*phase - 2*t + 1))/(2*(T - 1));
    y=(1*cos(dir)*(T + 2*phase - 2*t + 1))/(2*(T - 1));
    z=0;
end
end

```

Figure 3.14. New Feet_trajectory function.

Note: it's important to stress that the parabola is now in the z-y plane, and not in the x-y plane like the function "*parabola2.m*"

3.5 Adaptive Trajectories Method

Once the robot is able to perform parabolas with its feet, it is time to determine the method of adaptive trajectory generation.

The method proposed in this work consists of performing a parabola for the robot's foot. This parabola does not suffer any change if no obstacle is found. However, if the sensor on hexapod's foot detects any kind of obstacle, the leg stops its movement. In

this moment, robot's body starts advancing over the obstacle. This way, robot can overcome obstacles without a need of a camera or any other sensors.

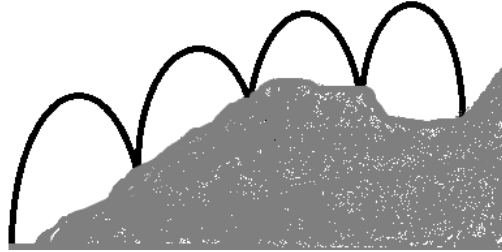


Figure 3.15. Feet_trajectory function.

Figure 3.15, shows a schematic way how robot's foot should move over irregular terrain. As it can be seen, robot would pass both over higher obstacles and over lower obstacles (holes).

In order to keep on advancing with the function that makes this method possible, it is important to check in order to see if the code will work properly and without singularities or failures.

With the purpose of checking if the function is correct, it was obtained the step length from the z coordinate.

For this, it is only necessary to clear the *step-length* from the function and displayed. As it can be seen, it always provides the value entered previously, without any indetermination or failure.

$$z = -\frac{(a*y)^2}{4} + stepheight \quad (11) \quad \rightarrow \quad stepheightttest = z + \frac{(a*y)^2}{4} \quad (12)$$

$$a = 2 \frac{\sqrt{stepheight * 4}}{Step\ length} \quad (13)$$

$$steplengthfunctionfz = \frac{2\sqrt{4*stepheightttest}}{a}. \quad (14)$$

The function would remain as shown in figure 3.16.

As it can be seen when code is executed, the values are the expected ones.

```

function [x, y, z] = feet_traj(phase, T, l, h, t, dir)
%if current time is less or equal than legs phase

a=2*sqrt(h*4)/l;

if(t<=phase)
    x=-(1*sin(dir)*(T - 2*phase + 2*t - 1))/(2*(T - 1));
    y=-(1*cos(dir)*(T - 2*phase + 2*t - 1))/(2*(T - 1));
    z=0;
end
%if current time is more than legs phase and less that phase+1
if(t>phase) && (t<=phase+1)
    x=(1*(t-phase-1/2))*sin(dir);
    y=(1*(t-phase-1/2))*cos(dir);
    z=-((a*y).^2)/(4)+h;

    %Calculation of steplength as a function of z
    htest=z+((a*y).^2)/(4)
    lfunctionofz1= (2*sqrt(htest*4))/a
end
%if current time is more than legs phase+1
if(t>phase+1)
    x=(1*sin(dir)*(T + 2*phase - 2*t + 1))/(2*(T - 1));
    y=(1*cos(dir)*(T + 2*phase - 2*t + 1))/(2*(T - 1));
    z=0;
end

```

Figure 3.16. New Feet_trajectory function. Check of singularities.

3.6 Obstacles

First approach

The next step for the development of adaptive trajectories would be to input the height of the obstacle, plotting a parabola that stops when reaches the obstacle.

The first problem we need to overcome is that our current function is based on time. It has 3 “if” cases depending on time. However, for an adaptive trajectory, the legs movement cannot depend on time but on the characteristics of terrain (x , y , z) coordinates. For this reason, a completely different function (with the same expression for the parabola) was built. This function generates different coordinates depending on z coordinate. The idea is to build a parabola, but in the very moment it reaches an obstacle, start moving the leg backwards.

The trajectory script is the following:

```

y=[];           %Inizialization of Vectors
z=[];

a=2*sqrt(step_height*4)/step_length;

y(1)=-step_length/2; %Inizialization of y vector
i=2;                %Index

for t= 1:0.1:70    %Vector of times

if(y(i-1)==-step_length/2) || (obstacle==0)

    y(i)=y(i-1)+0.01; %Y vector. Linear
    z(i)=-((a*y(i)).^2)/(4)+step_height; %Parabola

    %if reached the obstacle (precision 0.1)
    if(abs(obst_height-z(i))<0.1) && (y(i)>0)
        obstacle=1; %Signal of obstacle (true/false sensor in
legs)
    end

    i=i+1;
end

if (obstacle==1) && (y(i-1)>-step_length/2) %obstacle found

    %Backwards movement

    z(i)=obst_height;
    y(i)=y(i-1)-0.1;

    i=i+1;
end

plot (y,z)

```

Figure 3.17. Trajectory script.

This function is an independent function, built apart from *hexapod.m* code, as a first step to check the suitability. It uses vectors to store values of coordinates, in order to plot them after.

This is what we get when execute it:

```
>> feet_traj_2(5,8,0,3)
```

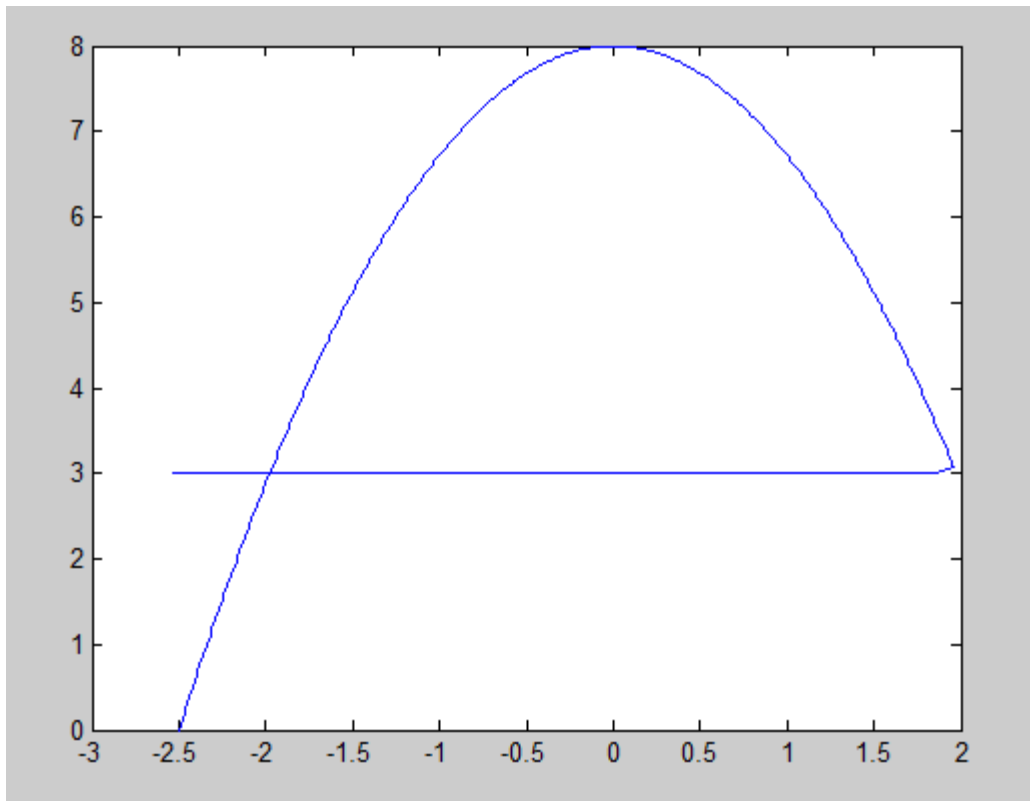


Figure 3.18. Leg trajectory shape in Z-Y frame.

As it can be seen in this first approach, when the parabola reaches the value 3 for z coordinate, it starts moving backwards, until value -2.5 ($-\text{step_length}/2$), which is exactly the shaped looked for. Since we are working with discrete values, there is a limit on the precision we can obtain. That is the cause of the little mistake when the obstacle is reached.

With this first approach made successfully, the challenge now is to adapt this function to the whole code, in order to simulate and start checking the robot movement.

Second approach

The first function developed, is basically the previous one, with some changes for adapting it to the *hexapod.m* code.

As it has already been said, in the function "*feet_traj.m*", the Y coordinates, generation is based on the variable " t ", this way, they kept advancing and there was no need to store the values. However, in this new function, the decisions are made based on the previous values of Y coordinates, so there is a need to "re-use" the previous values of Y coordinates in order to generate the next one. In addition, the code is much clearer if variable *obstacle* is not a local variable from "*feet_traj_4*" function but a global variable of "*hexapod.m*" code, and it is received from the function.

```

function [x, y, z] = feet_traj_4(phase, T ,step_length,
step_height,t,dir,obst_height)

%Phase and T not used

a=2*sqrt(step_height*4)/step_length;
obstacle=0;

if(t==0)
    y=-step_length/2 ;
end
    %Inizialization of y (only in first moment)

if(obstacle==0)

    y=-step_length/2+t*10;           %y vector. Linear
    z=-((a*y).^2)/(4)+step_height; %Parabola
    x=0;
    %if reached the obstacle (precision 0.1)
    if(abs(obst_height-z)<0.6) && (y>0)
        %Signal of obstacle (true/false sensor in legs)
        obstacle=1
    end
end

if (obstacle==1) && (y>-step_length/2) %obstacle found

    %Backwards movement

    z=obst_height;
    y=y-0.1;
    x=0;

```

Figure 3.19. Adaptive trajectory. Second version.

For that, the new concept of the function is the following.

This way, if there is no obstacle, the movement is the regular parabolic one, however, if there is an obstacle, z values are equal to *obstacle_height* and Y coordinate starts to move back.

It is important to note that variable “*obstacle*” is only used in this MATLAB® simulation, in order to make it work, but in the actual robot, it will be the signal from the sensors on the legs.


```

function [x, y, z] = feet_traj_4(phase, T, step_length,
step_height, t, dir, yp, obstacle, obstacle_height)

%Dir is ot used

a=2*sqrt(step_height*4)/step_length;

if (t==0)
yp=-step_length/2 ;
end

if (obstacle==0)

%Forward movement. Parabola

y=yp+0.2; %y vector. Linear
z=-((a*y).^2)/(4)+step_height;
x=0;

end

if (obstacle==1) && (yp>-step_length/2) %obstacle found

%Backwards movement

z=obstacle_height;
y=yp-0.2;
x=0;
end

```

Figure 3.20. Adaptive trajectory. Second version.

For that, outside the function (in *hexapod.m* code) there is a need to establish if the z coordinate reached an obstacle, in order to set *obstacle* variable equal to 1. For that, it was built this new part of code, (outside the function) (figure 3.21).

The results for this new function seem to be correct, and the MATLAB® model for the robot moves as desired.

```

%if reached the obstacle (precision 0.6)
if(abs(obstacle_height1-RF_z)<0.4) && (RF_y>0)
    %Signal of obstacle (true/false sensor in legs)
    obstacle1=1;
end

if(abs(obstacle_height2-RM_z)<0.4) && (RM_y>0)
    obstacle2=1;
end

if(abs(obstacle_height3-RH_z)<0.4) && (RH_y>0)
    obstacle3=1;
end

if(abs(obstacle_height4-LF_z)<0.4) && (LF_y>0)
    obstacle4=1;
end

if(abs(obstacle_height5-LM_z)<0.4) && (LM_y>0)
    obstacle5=1;
end

if(abs(obstacle_height6-LH_z)<0.4) && (LH_y>0)
    obstacle6=1;
end

```

Figure 3.21 . Additional code to “hexapod.m”.

However, it seems to fit better if the second buckle is guided by coordinates (space instead of time). Thus, the new second buckle is the following one:

y=-step_length/2:res:step_length/2.

This way, it will generate values for *Y* coordinates, no matter how long does it take for it.

Third approach

After checking the previous function, it came up a necessity to make some changes in order to stop generating coordinates after reaching the other side of the parabola. This is the current step shape:

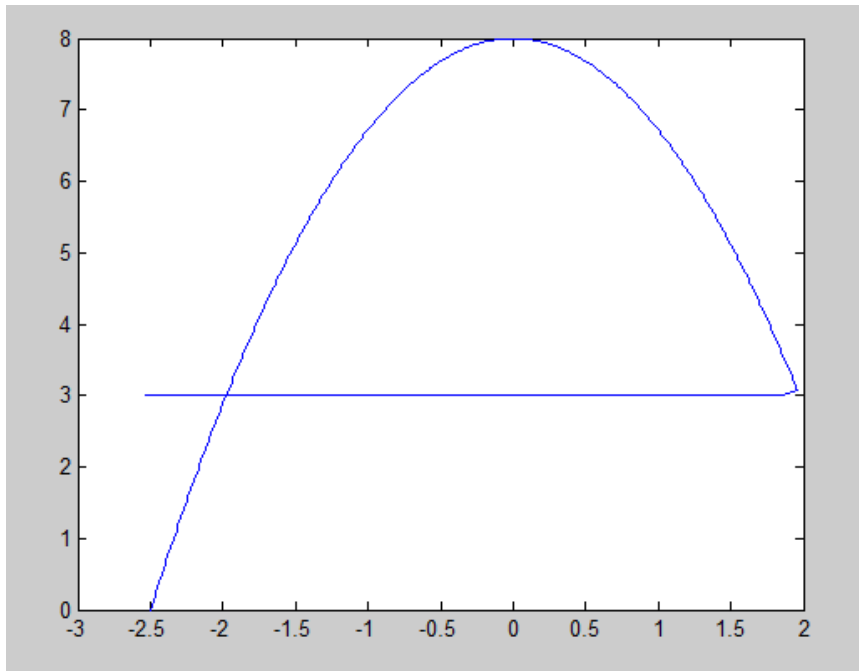


Figure 3.22. Current step shape. Z-Y frame.

However, the desired step shape is the following.

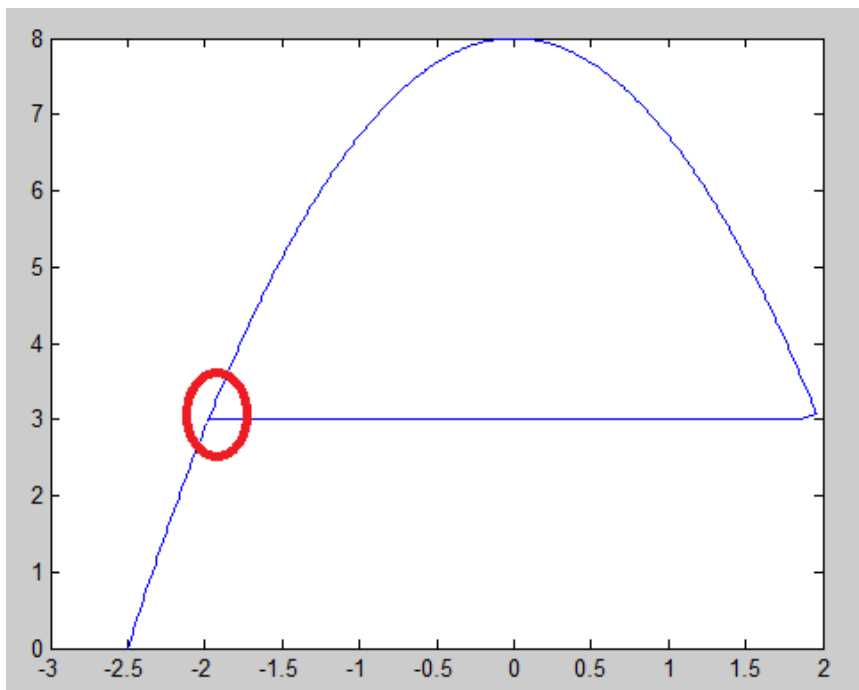


Figure 3.23 . Desired Step shape.

To accomplish this, it can be done through two different ways:

1. Recalculating the step length through mathematical expressions.

2. Adding a new condition to three second parts of the function, with the purpose of stopping the coordinate generation when the backwards movement reaches the other side of the parabola

The second option is chosen, for being the simplest one.

Before writing any piece of code, there is some mathematics that needs to be done. The Y value that needs to be found is the one in which the line crosses the parabola (stressed with red in the picture). If this value is found, the new condition for the parabola becomes easy to write.

Starting with the equation of the parabola:

$$z = -\frac{(a*y)^2}{4} + stepheight, \quad (15)$$

$$\rightarrow (obstacleheight - stepheight) * 4 = -(a * y)^2, \quad (16)$$

$$\rightarrow \sqrt{-(obstacleheight - stepheight) * 4} = a * y, \quad (17)$$

$$\rightarrow y = \frac{\sqrt{-(obstacleheight - stepheight) * 4}}{a}, \quad (18)$$

$$\rightarrow y = \frac{\sqrt{-(obstacleheight - stepheight) * 4}}{a}. \quad (19)$$

Once this value is found, the new function can be constructed.

```
function [x, y, z] = feet_traj_4(phase, T, step_length,
step_height, t, dir, yp, obstacle, obstacle_height)

%yp is the previous value of y coordinate.
%Phase and T are not used

a=2*sqrt(step_height*4)/step_length;

if(obstacle==0)
    y=yp+0.2; %y vector. Linear
    z=-((a*y).^2)/(4)+step_height; %Parabola
    x=0;

end

if (obstacle==1) && (yp>-sqrt((step_height-obstacle_height)*4)/a)
%obstacle found
    %Backwards movement
    z=obstacle_height;
    y=yp-0.2;
    x=0;
end
```

Figure 3.24. Adaptive trajectory generation function. Third version.

As it can be seen in the simulation, with this function, the step shape is the desired one.

Forth approach. Finishing the step

The next problem that needs to be solved is the method to finish the step. In the current function both of current conditions for coordinates generating will be false after finishing the movement. And this gives an error when the function is called without any parameter as an output.

In order to use the same function for every leg of the robot (increasing clearance), after the legs finish their movement, the function will keep generating coordinates, but they will be the same coordinates as the coordinates in the instant they stop. Thus, every leg of the robot will start its movement for each step at the same time, although they might finish at different time instants. (The leg that will find an obstacle will finish the movement before the leg that does not find any). This way, we can use the same function for every leg making the code more flexible and easier. Besides, the movement will be neater.

Fifth approach. Different obstacles for each leg

The purpose of this robot is to be able to walk over any surface, adapting itself to the terrain. For this reason, it seems natural, that in our simulation code there should be different obstacle heights and different obstacle signals. This way, we will make each leg movement independent, so the robot will adapt to the terrain.

The first thing to do for this is to set different obstacle heights, so the robot can decide, for each leg, when this obstacle appears.

And it is not enough with this, since in the actual robot there will be moments when no obstacle is found. To make it the more similar to the actual robot as possible, if there is no obstacle, obstacle height should be set as 0, and in the part of the code that generates the variable, instead of *obstacle=1*, we should change it to *obstacle=0*.

After this, the feet trajectory function needs to be modified. This is the new function, which is now able to handle different movements for each leg, depending on whether there is an obstacle or not.

```

function [x, y, z] = feet_traj_4(phase, T, step_length,
step_height, t, dir, yp, xp, zp, obstacle, obstacle_height)

%xp, yp, and zp are the previous values of the coordinates.
%Phase and T are not used
a=2*sqrt(step_height*4)/step_length;

if (obstacle==0) && ((t==--step_length/2) || zp>0)

    %There is no obstacle. We are in the first iteration of the loop
    or we are making the parabola movement (zp>0)

    y=yp+0.3; %y vector. Linear. 0.3 is the minimum speed. If the
speed is still too much, this function is the one to be modified.
    z=-((a*y).^2)/(4)+step_height; %Parabola
    x=0;

elseif (obstacle==0) && (zp<=0) && (yp>--step_length/2)

    %no obstacle was found and z has got 0 (parabola movement finished).
z maintains its position and Y starts moving back.
    z=0;
    y=yp-0.3;
    x=0;

elseif (obstacle==1) && (yp>-sqrt((step_height-obstacle_height)*4)/a)
%obstacle found

    %Obstacle found. Backwards movement. "-sqrt((step_height-
obstacle_height)*4)/a" is the crossing point between parabola and
line.

    z=obstacle_height;
    y=yp-0.6;
    x=0;

elseif (obstacle==1)

    %There was an obstacle, but the backwards movement has finished. It
generates the same coordinates.
    z=obstacle_height;
    y=yp;
    x=0;

end
end

```

Figure 3.25 . Adaptive trajectory generation function. Fourth version.

Figure 3.26 shows how different robot feet react to different obstacle heights, obtaining independence between legs.

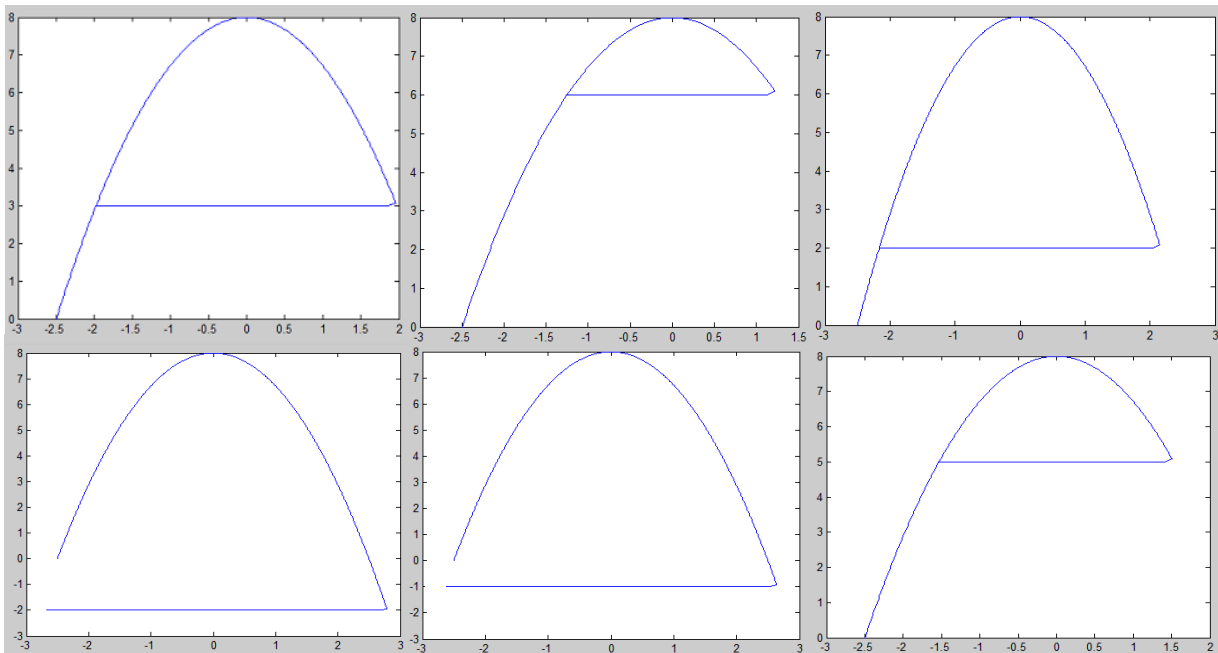


Figure 3.26. Different feet trajectories for different obstacles height.

Sixth approach. Tripod gait

The previous function worked perfectly. Yet, it had a problem. As it is previously explained, since the function keeps generating coordinates after the obstacle is reached, in the next step, all the legs would start its movement at the same time. This way, there is no option to program a specific gait.

In order to fix that, a new function was built, which, instead of generating the same coordinates, it sets the *obstacle* variable to 0, this way, when the leg finish its movement, it starts a new one, without waiting for other legs to finish. This permits to perform the Tripod gait, by programing different starting points for the legs. Legs are now completely independent, although they will maintain the tripod gait as long as they are moving. The function remains as follows:

```

function [x, y, z,obstacle] = feet_traj_5(step_length,
step_height,t,dir, yp, xp, zp, obstacle, obstacle_height)

%xp,yp, and zp are the previous values of the coordinates.
%Phase and T are not used
a=2*sqrt(step_height*4)/step_length;

if(obstacle==0)

    y=yp+0.3;
    z=-((a*y).^2)/(4)+step_height; %Parabola
    x=0;

elseif(obstacle==1)

    if (yp>-sqrt((step_height-obstacle_height)*4)/a) %obstacle found

        %Obstacle found. Backwards movement."-sqrt((step_height-
        obstacle_height)*4)/a" is the crossing point between parabola
        and line.
        z=obstacle_height;
        y=yp-0.6;
        x=0;

    else

        %There was an obstacle, but the backwards movement has
        finished. It generates the same coordinates.
        z=obstacle_height;
        y=yp;
        x=0;

        obstacle=2 % allows to set obstacle variable to 1 again.
    end
end
end

```

Figure 3.27. Adaptive trajectory generation function. Fifth and final version.

The code was checked in the simulator, obtaining a successful movement of the robot. However, the code now has one disadvantage. Since there is no possibility to know when one whole step will end (it depends on terrain characteristics), it is better to use a long-lasting for loop, that will be the whole length of the robot movement.

The robot moves now as desired, with a perfect tripod gait. The figure 3.28 shows a random instant of the simulation, where it shows different obstacle heights and how the robot adapts to them.

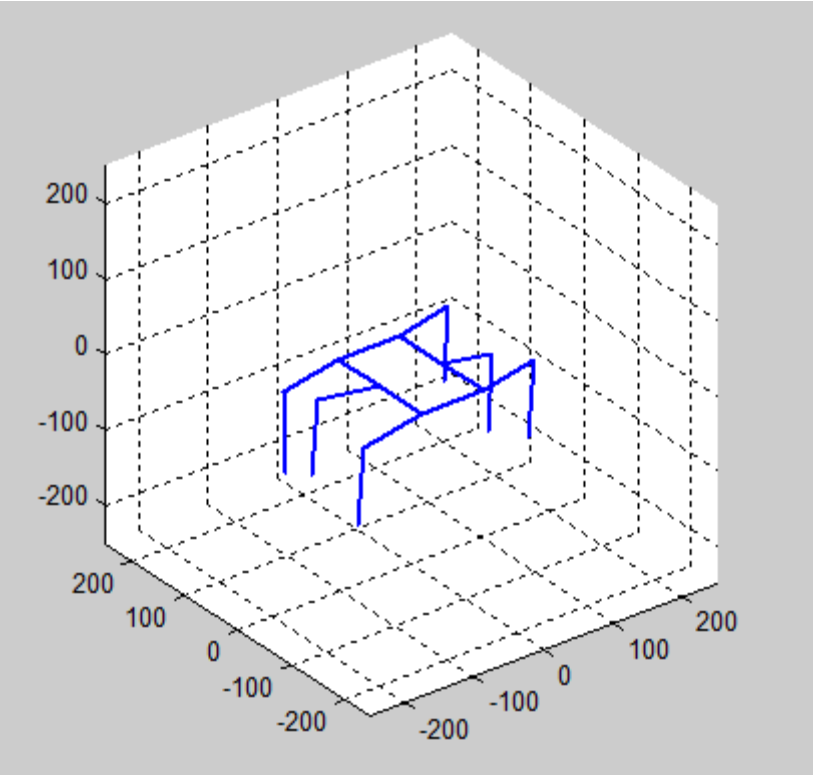


Figure 3.28 . Random instant of the simulation.

4. CONCLUSIONS

1. Parabola function is better for foot trajectory than sinusoidal functions, because it has the property of not changing concavity when zero is passed.
2. Developing independent parabolas for the different robot's feet and being able to stop its generation when any obstacle is reached, is a method that works properly on MATLAB® simulator, and it seems to be very appropriate for hexapod robots.
3. Legged robots are much more complicate to control, but the potential is bigger over wheeled robots, when talking about unknown surfaces and terrains.
4. Denavit-Hartenberg algorithm, together with geometric inverse kinematics method, easily allows to model the robot, and permit to develop a simple algorithm to control it.
5. As for the experiments carried out, it is clear that being able to test the trajectory generation functions in a software is an enormous advantage, since almost every parameter can be checked.
6. Robot movement was checked with the simulator, inputting different heights for obstacles encountered by different feet, and observing how robot adapted to them.
7. Robot movement reacts very well to different obstacle heights, since the method for trajectory generation makes robot's legs to move independently.
8. As a general conclusion reached, robotics is a wide field of study, where a lot of research is still to be done, especially concerning control systems.
9. Optical methods to examine unknown terrain (artificial vision) require more capacity for the microcontroller of the robot than yes-no sensors. The reason is that they need to process too much information, and this may affect real-time operations. On the other hand, using yes-no sensors, permit more speed in the operations. However, using yes-no sensors, have a big disadvantage, which is the necessity of knowing the maximum height of the obstacle. This height needs to be mandatorily less than step-height.

Future work

The function as it currently remains is not able to walk in a desired direction. Future work on trajectory generation should include some functions that would permit the robot to walk in every direction desired. The easiest way to achieve this would be to include sinusoidal functions that would give the trajectory the desired direction.

In the present moment, there is no control over the speed of robot's foot. It would be logical to include some parameters that would permit to control robot's speed, in order to find the best speed for the different obstacles.

More gaits should be developed, in order to improve the flexibility of the robot.

5. REFERENCES

- [1] Mechatronic systems and materials. Edited by Andrejus H. Marcinkevicius and Algirdas V. Valiulis. Trans Tech publications.
- [2] D.M. Wilson, Insect walking, Annual Review of Entomology 11 (1966) 103–122.
- [3] Introduction to Autonomous Mobile Robots. Roland Illah, R. Siegwart, Nourbakhsh.
- [4] Gaits and energetics in terrestrial legged locomotion D.C. Kar a*, K. Kurien Issac b, K. Jayarajan a.
- [5] Paul, Richard (1981). Robot manipulators: mathematics, programming, and control : the computer control of robot manipulators. MIT Press, Cambridge, MA. ISBN 978-0-262-16082-7.
- [6] http://en.wikibooks.org/wiki/Robotics/Types_of_Robots/Wheeled.
- [7] http://no.cyclopaedia.net/wiki/SCARA_robot
- [8] <http://www.challenge.toradex.com/>
- [9] ATmega16 datasheet.
- [10] <http://en.wikipedia.org/wiki/Robot>
- [11] http://en.wikipedia.org/wiki/Cartesian_coordinate_robot
- [12] http://en.wikipedia.org/wiki/Cartesian_coordinate_robot
- [13] http://en.wikipedia.org/wiki/Parallel_manipulator
- [14] <http://en.wikipedia.org/wiki/Ro>
- [15] <http://www.allonrobots.com/>
- [16] <http://en.wikibooks.org/wiki/Robotics>
- [17] http://en.wikipedia.org/wiki/Spherical_robot
- [18] <http://www.thefullwiki.org/Wheeled>
- [19] http://www.openbook.com.vn/index.php?title=Robotics/Types_of_Robots/Wheeled