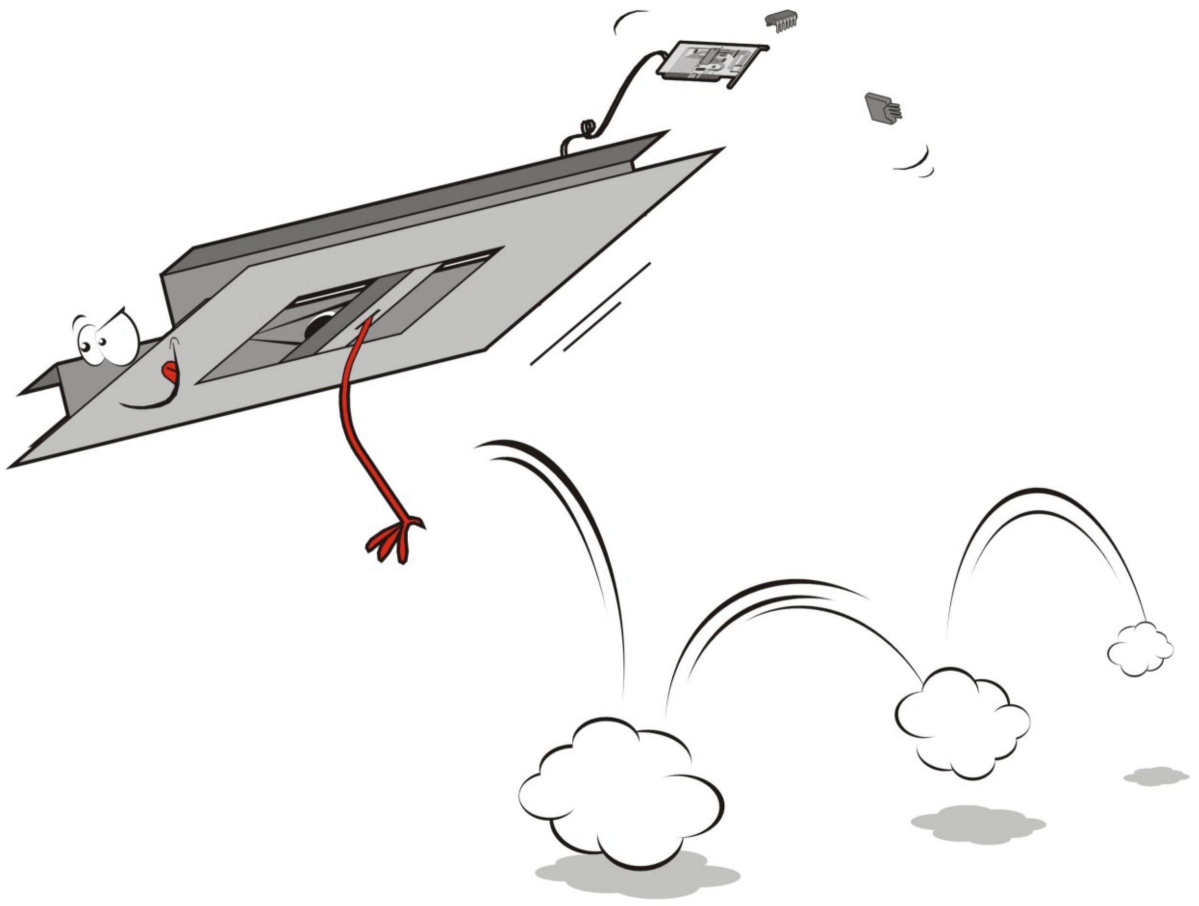
Official advisor at CMU: Ben Brown
Assistant at CMU: Garth Zeglin
Professor at CMU: Illah Nourbakhsh
Professor at EPFL: Roland Siegwart

Carnegie Mellon University, Pittsburgh, PA **Swiss Federal Institute of Technology, Lausanne, CH**

Diploma project

First Jumps of the 3D Bow Leg Hopper

Jean-Christophe Zufferey
Micro-engineering, EPFL
February 2001



Picture: courtesy of Losange Communication – www.losange.ch

Abstract

This project is about a simple control of a robot that has a single leg so hopping is the only gait it can use. The characteristics of this machine are:

1. The leg is bow-shaped and works like a spring that can be compressed between one step and the next in order to add energy.
2. The body attitude is passively stabilized during stance by the low placement of the center of gravity.
3. Onboard batteries electrically power all the actuators. No wire to outboard devices is required.

This 3D machine is a result and a generalization of an in-depth study of a preliminary planar prototype: the 2D Bow Leg Hopper. The motion of this early robot was restricted to the plane. A tether mechanism constrained the machine to move with just three degrees of freedom. Actually, the robot moved on the surface of a large sphere centered at the tether pivot. This planar monopod hopper demonstrated the efficiency of this new type of running robot.

The hip joint of these machines is attached to the body slightly above the center of mass. Thus, simply decoupling the leg and the body during floor contact passively stabilizes the body, which is subjected to natural pendulum forces. As a result, the body will swing fore and aft during hopping.

From a control point of view, it is necessary to know the body attitude because the controlled leg angle is specified in the body frame but the useful angle is the one between the leg and the floor at touchdown.

Although it was quite straightforward to measure the body angle with the planar prototype (a potentiometer was simply mounted between the boom and the robot), recovering the body attitude of the 3D machine is trickier. Thus an important part of this project is about sensing.

Four optical range finders have been mounted below the body. A filter is implemented on the fast onboard 8-bit microcontroller that processes the asynchronous signals from the four distance sensors in order to compute the roll and pitch angles.

When moving from two to three dimensions, another complication appears. On the 2D prototype, the hip joint was simply a hinge. With the 3D machine, it becomes a gimbal-type hip thus complicating the mechanics and the control.

The scope of this project is the first step in the direction of the total 3D freedom: testing the hardware, designing and assessing the sensing system. To reach these objectives, a simple controller has been developed that tries to keep the leg vertical with respect to the floor. A human operator can add – through a radio command – a small offset to the control output in order to compensate for the errors or disturbances. First results are the product of experiments conducted with the 3D Hopper in a reduced-gravity configuration.

Contents

1 Introduction

1.1 Official project abstract

The Bow Leg Hopper is a new type of running robot with an efficient, flexible leg. A one-legged planar prototype [Zeg99] has been developed that passively stabilizes body attitude and is efficient enough to use on-board batteries. This prototype, the 2D Bowleg Hopper, has demonstrated crossing of simple artificial terrain including stepping stones and shallow stairs.

The 3D Bow Leg Hopper operates on the same design principles as the 2D Hopper: it uses an efficient spring for a leg that is freely pivoted at the hip and positioned during flight by control strings, and has the center of mass balanced below the hip. The chief difference is that the leg must freely pivot in two degrees of freedom, which substantially complicates the mechanical design.

This second prototype has just been assembled and the problem is now sensing body attitude (and velocity) over a flat floor. The diploma work consists in getting the 3D Hopper to the point where it could hop stably in place, or at least stabilized enough that a human driver is able to roughly control its position.

1.2 A few words about the recent history of the legged machines

A major motivation for studying the principles of legged locomotion is to develop useful legged vehicles, which should be able to move on rough terrains that are unreachable with conventional wheeled or tracked vehicles. Such scientific research will also lead us to a better understanding of human and animal locomotion.

The scientific study of legged locomotion began about a century ago, in 1878, when Muybridge published his stop-motion photographs of over forty mammals in *Scientific American.* A very good overview of the walking and running machines can be found in [Rai86], p. 6, that covers the 108 years between this first attempt and the publication of the Raibert's book in 1986. During this period, none of the built machines was capable to really run[1] until 1980.

Pioneering work in the field was done by Raibert at the Leg Lab [wwwLegLab], first at Carnegie Mellon University and then at MIT, which produced a series of running robots. The first prototype was a pneumatically actuated planar[2] monopod. Following was a hydraulically actuated 3D monopod. This latter machine, built in 1983 at CMU, will constitute our reference and object of comparison because no other fully three-dimensional one-legged hopping machine has been developed so far[3]. A comparison between the 3D Bow Leg Hopper and this former 3D one-legged hopping machine will take place in § 2.1.2.

After that, a lot of multi-legged and/or planar running robots followed. A condensed description may be found in [Zeg99] under § 7.1.1, p.127. This PhD thesis is freely downloadable from [www2DBowLeg].

Running is a process of falling, storing energy and rebounding. The leg-ground forces only determine the overall movement thus involving very dynamic control. The major difficulties in the

[1] Where running implies at least a moment of ballistic motion during which no leg or part of the machine is in contact with the floor.
[2] The adjective 'planar' will be used in this report to describe machines that are attached to a fixed point of the laboratory by a boom thus reducing the degrees of freedom from six to three. This means was often used in order to reduce the complexity of three-dimensional running in former experiments. The '2D' prefix will also often be used in the case of planar machines.
[3] With the exception of Robop that is a self-stabilizing hopping robot built in 1996 at the Leg Lab (MIT). It has no active electronic sensors to use for feedback. More information may be found on the Leg Lab web site [wwwLegLab].

realization of such legged machine are the balance, the lightweight actuation, the power requirements and the environment sensing. The first one has been studied by a large number of researchers such as Raibert or McMahon [McM84]. But very often, the built machines avoided the difficulty of on-board power location and lightweight actuation by providing an umbilical cable to supply for energy and so allowing for more heavy actuators. Concerning the huge problem of environment sensing, a lot of work is still to do before being able to build legged machines, which could try to outdo animal behavior in rough terrains.

More recently, 3D hopping machines have been developed in a completely new way by researchers at the Department of Energy's Sandia National Laboratories [wwwSandia]. They use combustion-driven pistons to make leaps as high as 20 feet. Although this research is very promising for long distance displacements, there is no possible comparison between these machines and our 3D Bow Leg Hopper since they do not use active balance to run but 'simply' jump very high, fall on the ground, lengthily recover the orientation and jump again.

1.3 This project

1.3.1 Context

When Prof. Illah Nourbakhsh told me about this hopping robot, I found this project very exciting maybe because I had no idea about how we could manage to make this crazy machine stay upright. I didn't know anything about the long story of the legged machines and the glorious past of CMU in this domain.

Two scientists, here at CMU, did a lot of work that strongly influence my project. First Ben Brown, my official advisor, had the idea of a bow-shaped leg for hopping robots. Then Garth Zeglin, my assistant, accomplished a PhD about a planar version of the Bow Leg hopper (Fig. 1.1). This work demonstrated the efficiency and natural stability that makes self-contained running robots feasible. The 2D Bow Leg Hopper was able to cross simple uneven terrain in the laboratory using a graph search planner [Zeg99].

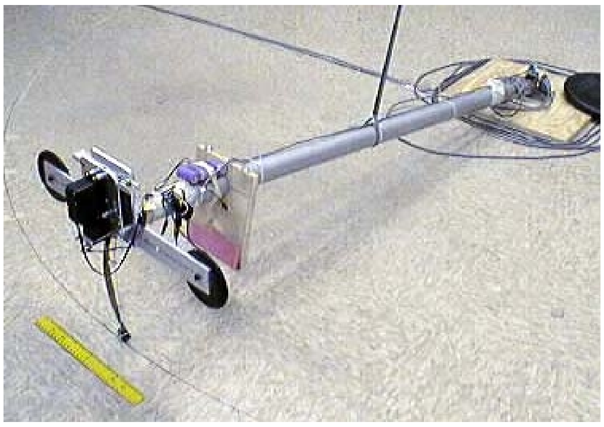


Fig. 1.1 from [Zeg99] – The 2D Bow Leg Hopper

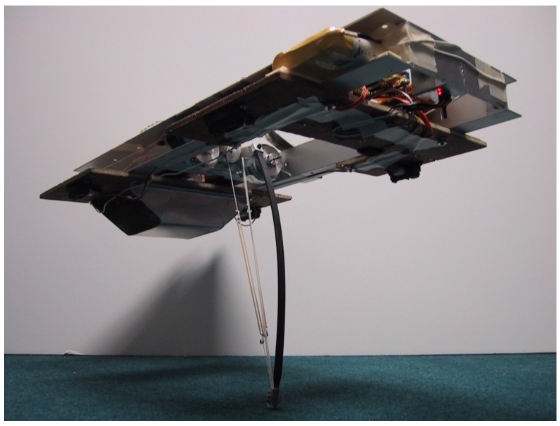


Fig. 1.2 – The 3D Bow Leg Hopper

After the end of his PhD, Garth continues his research in the direction of the ultimate goal of this whole work: "the development of a fully autonomous running machine that may bound its way across rugged terrain". He designed, in collaboration with Ben, the core of the current 3D Bow Leg Hopper (Fig. 1.2): the gymbal-type hip and the complex system of pulleys that allow the control and the bending of the leg (see § 2.1). Garth also carried out some preliminary work concerning the embedded microprocessor system.

When I began with this project, the core mechanism for leg positioning was realized. And Ben was working on the design of the thrust mechanism and the body of the robot. During the whole length of my work, he took the responsibility for developing the mechanics. On his side, Garth helped me to make the right choices from the control point of view and did also a great job in designing the actual electronic board.

As a reminder, the 3D Bow Leg Hopper operates on the same design principles as the 2D Hopper. It uses an efficient spring for a bow-shaped leg that is freely pivoted at the hip and positioned during flight by control strings. The center of mass is balanced below the hip thus allowing passive stability of the body.

1.3.2 A first step in the direction of full 3D freedom

This project is a first attempt to generalize the experience with the planar hopper to a three-dimensional machine. So the scope of my work is essentially:

1. Designing and assessing a sensing system.
2. Developing a simple controller.

As a first step in the direction of 3D freedom, these developments shall allow us to test the new hardware and accomplish some preliminary experiments to see how the Bow Leg functions with additional degrees of freedom.

Although the main part of the development described in [Zeg99] generalizes well to the three-dimensional case, a set of basic differences between the 2D and the 3D prototypes remains:

1. Sensing body attitude:

   Although it was quite straightforward to measure the body angle with the planar prototype (a potentiometer or an optical encoder was simply mounted between the boom and the body of the robot), recovering the body attitude of the 3D machine is trickier.

2. The hip joint and positioning yoke:

   The leg must freely pivot in two DOF (instead of one in the case of the 2D Hopper), which substantially complicates the mechanical design and the control. The use of a gimbal joint is needed instead of a simple hinge joint. Like the leg, the positioning yoke also needs 2 DOF, which implies the use of at least three control strings. For more preliminary details, see Appendix C in [Zeg99].

3. Location of the controller:

   The planar hopper was controlled by an off-board PC using off-the-shelf I/O cards. As one of the goals of the new 3D prototype is to preclude any umbilical cable, the control must run in an onboard processor, implying processing power and speed limitation. The employed 8-bit micro-controller is an Ubicom SX52BD that runs at 50Mhz and doesn't have division or multiplication in its basic instruction set.

4. Body attitude stability:

   As Garth points out in his thesis, one of the important questions is about the generalization of the passive stability of the body: "The limited experience with the air-table planar prototypes suggests that the attitude stability may be marginal without the damping of the boom pitch bearing. This is expected to make passive attitude stability more difficult and active hip positioning may be required."

The controller I choose for these first 3D experiments 'simply' tries to keep the leg vertical with respect to the floor. This solution is only marginally stable by itself, however, on top of this control, a human operator can add, with help of a radio command, a small offset to the leg angle in order to compensate for the errors or disturbances. This relieves the human operator of the difficult task of estimating body orientation within the short 200ms falling time.

In general, the hopping control requires positioning the leg in world coordinates but it is controlled in body coordinates. The stability of a monopod hopping machine is quite sensitive to small angular errors in leg position. Further, the on-board sensing of three-dimensional orientation and location from a moving platform is very delicate. Inertial sensors do not work well until continual impulsive forces. Vision is a likely future candidate, but will probably require multiple cameras and certainly a lot of computation. What remains is an arrangement of distance sensors such as sonar or laser range finder (structured light, triangulation), which may work well on quite flat surfaces. Therefore I take the compromise decision to see how much I can do with low-bandwidth but lightweight and cheap optical rang sensors, focusing first on body orientation.

As my project represents a very initial incursion in the world of self-contained three-dimensional one-legged hopping machines, a number of problems have not been addressed. Some of them are listed here:

1. The control and the measurement of the lateral velocity. The former part is one the main goals of [Zeg99]. The latter represents a great difficulty for unconstrained 3D hopping machines. This problem has been addressed by Raibert in [Rai86], p. 69, under "Estimate Forward Velocity from Leg Motion".
2. The control of the yaw rotation. Some explanations about this problem appear in [Zeg99], p.17 and in Appendix C. On his side, Raibert argues that his machine "has no front or back, and the leg can move about equally well in all directions. […] control of heading does not have to take the facing direction of the machine into account – steering does not require turning." [Rai86], p.66.
3. Energy stored in the leg on each hopping cycle. A lot of considerations about this can be found in [Zeg99]. For the first experiments with the 3D Bow Leg Hopper, a constant amount of energy is injected into the system during each bounce cycle.

In order to somewhat reduce the difficulties the first experiments are conducted in a reduced-gravity configuration. See § 6.1 for a description of the employed system and its consequences on the robot behavior.

1.3.3 Organization of the report

This report is organized as a progression from mechanical design to control considerations including a detailed description of the sensing system.

Chapter 2 presents the hardware. Although I am not the designer of the 3D Bow Leg Hopper, an overview of the mechanical structure of this machine is needed in order to understand the following chapters. The global geometry shall be described as well as the sensors and the actuators.

Chapter 3 discusses the software and electronics domain, which is more representative of my contribution to the project. The employed microprocessor and the electronic board will be presented in this section. The need of monitoring and recording data from the robot during hopping experiments will also be explained.

Chapter 4 gives an in-depth coverage of sensing the body attitude, which represents the chief part of my work. Because sensing is of primary importance and much more difficult than in the 2D case a full chapter is devoted to it.

Chapter 5 follows with a description of the simple implemented controller. Although this controller is a reduced version, some comparisons will be made with the previous realizations in this domain: the famous Raitbert's three-part control and the linear controller implemented on the 2D Bow Leg Hopper.

Chapter 6 shows the results of the first laboratory experiments.

This report has a quite open end because the experiments with the real robot will continue after the hand-in date. Therefore, a lot of information such as videos are available on Internet [www3DHopper] and will be updated as often as possible. In the same idea, instead of inserting a lot of appendixes at the end of the report, I will put all the information that is self-contained on my web site.

In general and because that was the purpose of the planar prototype, a lot of results will be directly reused from [Zeg99]. However, for sake of clarity, some equations and figures are reprinted in the present manuscript.

For an easier understanding of the references indicated within brackets in the text, the books and papers are indicated by a compilation of the initials of their authors and the publication year. The web sites are referenced with a ‘www’ prefix. See Appendix 8.3 and 8.4.

2 Hardware

2.1 Overview

The goal of this chapter is to present the Bow Leg Hopper project. The hardware was not my main concern throughout this project but I think it is useful for the reader to have a quick overview of the machine before tackling its control and sensing system. Of course, a lot more in-depth information is available in [Zeg99].

2.1.1 Key ideas

The *bow leg* is a curved leaf spring attached to the body of the robot through a freely pivoting hip. A *bow string* linking the top of the leaf with the foot holds the leg in compression. Its length can be varied by the thrust mechanism (see § 2.3.4), allowing storing potential energy in the leg by bending the leaf spring.

In the design of the Bow Leg Hopper, a single curved leaf spring provides the leg structure, elasticity and energy storage. The spring and the hip bearing carry the high forces at ground impact. No actuator must support these shocks. This addresses four problems central to dynamic legged locomotion [Zeg99]:

1. A low-power actuator may be used for thrust by storing energy in the leg during flight.
2. Low-power actuator may be employed to position the leg during flight.
3. The free hip minimizes body disturbance torques.
4. The hopping cycle is energy efficient since negative work is eliminated and the spring has high restitution.

The machine is fully controlled using only actuation during flight. The leg is positioned, the *bow string* retracted to store the potential energy that will be automatically released during stance.

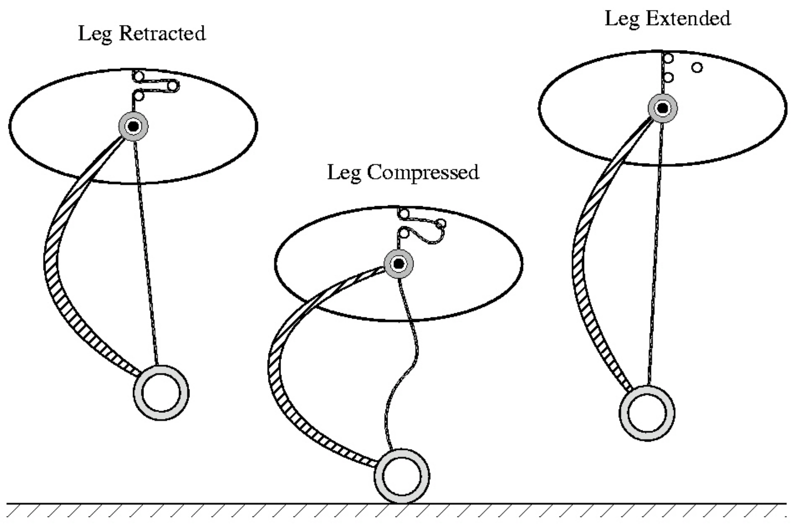


Fig. 2.1 from [Zeg99] – Three phases of the Bow Leg Hopper thrust mechanism

The three DOF of the leg are controlled using two *control strings* and the *bow string* itself. The *control strings* link the foot to the body through a quite complex system of pulleys (see § 2.3.2) and are controlled by two hobby servos. During stance, all the strings become slack and the hopper bounces passively off the ground with almost no forces or torques supported by actuators.

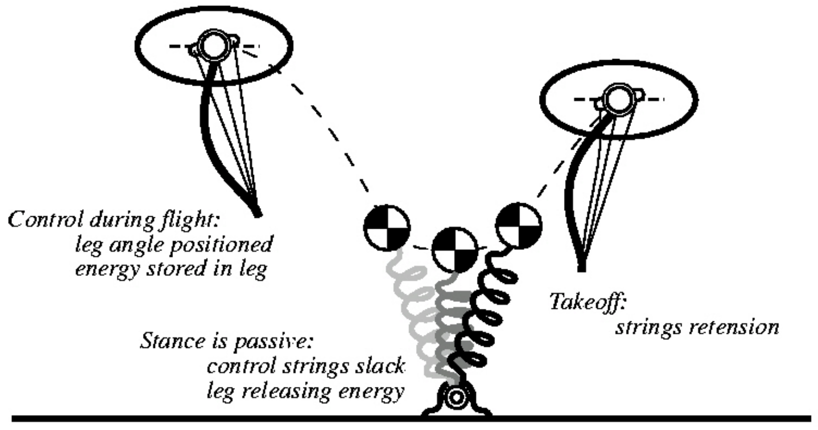


Fig. 2.2 from [Zeg99] – During stance, the hopper is a passive spring-mass system

The hip joint is attached to the body slightly above the center of mass so the body effectively hangs from the hip during ground contact allowing the natural pendulum forces, passively stabilizing the body attitude.

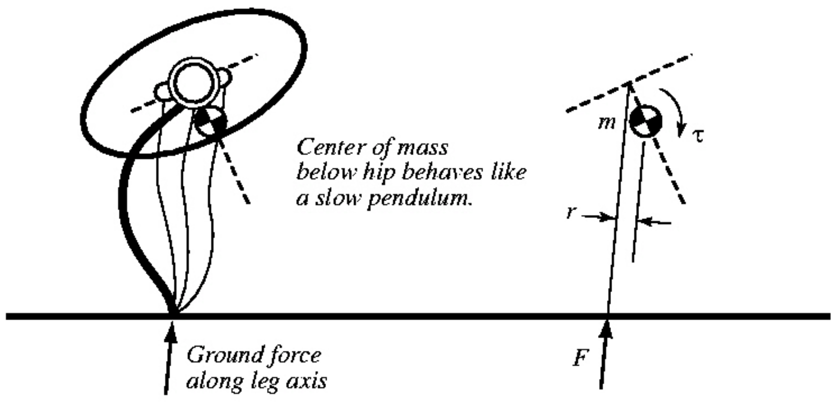


Fig. 2.3 from [Zeg99] – The center of mass below the hip allows passive body stability

These principles address the typical losses of legged systems: negative work and leg sweep [Ale90]. Moreover, since this mechanism passively stabilizes the body attitude, the model can neglect the body orientation. The physics of this model is very much like a stiff ball bouncing on a surface whose slope can be controlled at each impact [Zeg99], ch. 3.

Since the machine is passive during stance, the transition from one trajectory to another is determined by the leg angle and stored energy at touchdown. So trajectory control is performed by selecting these parameters during flight.

Based on this philosophy, the 2D Bow Leg Hopper has been the first hopping robot with on-board battery power. Following the same approach as Raibert in 1980, we would like to generalize the model to the three-dimensional case and apply it to the 3D Bow Leg Hopper.

2.1.2 Comparison with Raibert's 3D one-legged hopping machine

As said in the introduction Marc H. Raibert - helped by Ben Brown for the mechanics - pioneered the field of jumping robots [Rai86]. The first machine described in his book is a planar hopper followed by a hydraulically actuated 3D monopod based on the same kind of mechanics and control. The latter is the only machine comparable with the 3D Bow Leg Hopper, at this time. It has a gimbal joint hip that permits the leg to swing sideways with respect to the body, as well as fore and aft. The leg is a pneumatic cylinder whereas the pair of actuators that position it are hydraulic.

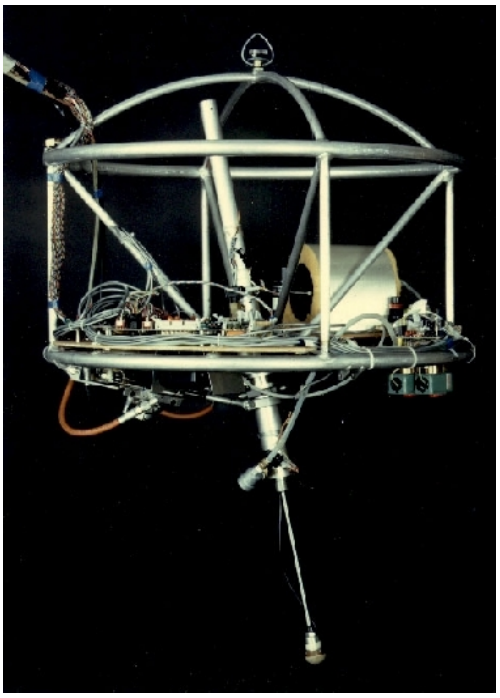
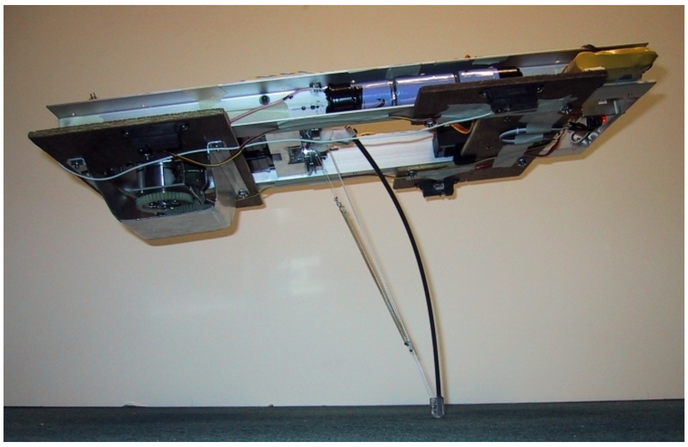



Fig. 2.4 – Left: Raibert's one-leg hopping robot from [wwwLegLab] - Right: the 3D Bow Leg Hopper

The chief differences between this machine and the 3D Bow Leg Hopper are the followings:

- The body attitude is actively controlled during stance thus complicating the control and the model but allowing a better control of the body attitude.
- The needed power is not delivered by onboard sources. An umbilical cable links the robot to electrical, pneumatic and hydraulic sources.
- The leg is a telescoping spring and not bow-shaped. It is also springy because air trapped in the leg actuator compresses when leg shortens.

As Garth points out in its thesis: "The Bow Leg Hopper is *mechanically programmed* during flight to set the initial conditions for impact. Any control must take place once per hopping cycle. High bandwidth control is eliminated…". This is not the case for the Raibert machine, which actively controls the torques applied to the body during stance.

One of the most important achievements of this work - both on the primary 2D monopod hopper and on the 3D version – is the decomposition of hopping control into three separate parts [Rai86]:

1. One part controls hopping height by delivering a certain leg thrust during each hopping cycle.
2. A second part of the control system regulates the forward rate of travel by placing the foot a specified distance in front of the hip as the machine approaches the ground on each step.
3. The third part of the control system corrects the attitude of the body by servoing the hip during stance.

The control of the Bow Leg Hopper can be achieved in a very similar way [Zeg99]:

1. The leg retraction at impact is similar to leg thrust. The thrust controls total energy, roughly equivalent to hopping height.

2. The touchdown leg angle is analogous to foot placement. The leg angle controls forward speed.
3. Since the body attitude should be passively stabilized, there is no counterpart.

This is a simple controller that is sufficient for holding a position or hopping at constant velocity. Under the light of this decomposition, my work focuses on half the requirements for a good achievement of the second part of the control. Actually, in order to set the correct leg angle at touchdown, the robot needs first to know the body attitude and then to sense the body lateral velocity. The first part of the control is simplified by storing a constant amount of energy into the bow leg.

Finally, as you can deduce from this brief comparison, if the 3D Bow Leg Hopper is able to control its lateral velocity - which is out of the scope of my project - it will become the first fully self-contained one-legged hopping robot.

2.2 Global geometric description

Here will be introduced some useful symbols and values. Please refer to Appendix 8.1 for a complete listing.

➢ **Frames definition**

The world frame is called *W* and the z-axis is perpendicular to the floor, in upward direction.

As the global position of the robot and its movement around the yaw axis are not under the scope of this project, a reference coordinate system, named *R*, has been chosen such as its center is at the location of the hip and the x- and y-axis define a plane parallel to the floor (x is the roll axis and y the pitch axis) and z-axis is always vertically oriented.

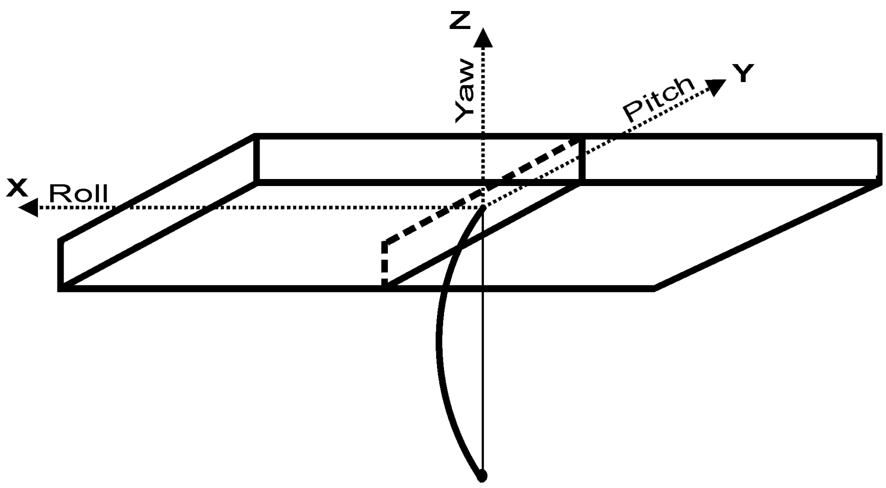


Fig. 2.5 – Bottom 3D view of the robot; definition of the main axis

A second frame called *B* is linked to the body of the Hopper. Its origin is the same as *R*. x- and y-axis are in the body plane and z is perpendicular to this body. At initial position, frame *B* coincide with frame *R*.

➢ **Angles definition**

The useful angles are defined in the following two-dimensional drawing (see Fig. 2.6). They are then suffixed with an r (roll axis) or a p (pitch axis) to be generalized to our three-dimensional case. When these suffixes are omitted that means the equation can be applied to both angles without restriction of the generality.

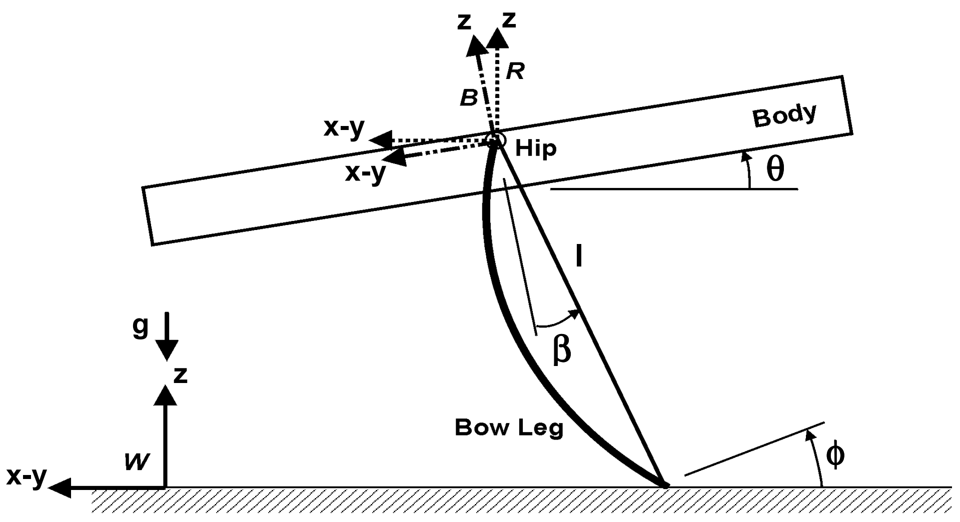


Fig. 2.6 – Angles definition

The body attitude is defined by θ_r and θ_p. θ_r is the angle between ${}^{R}y$ and ${}^{B}y$. θ_p is the angle between ${}^{R}x$ and ${}^{B}x$. These angles[4] will be estimated by the distance sensors (see ch. 4).

β_r and β_p are the leg angles with respect to *B*. Actually they represent the values that can be commanded by the controller.

The useful (the one we have to indirectly control) angle between the leg and the floor is determined by ϕ_r and ϕ_p.

The relationship between these angles is:

$$\phi = \theta + \beta \qquad \textit{Eq. 2.1}$$

- **Some dimensions**

The height of the hip above ground is designed by h. Note that as the center of mass (COM) of the body is very near below the hip (less than 1cm), the COM and the hip will often be taken as co-incident[5]. Of course, the height is time dependent: h(t).

As shown on Fig. 2.5, l is the initial leg length (before impact). The maximum leg compression during stance is denoted Δl.

2.3 Mechanics

2.3.1 Bow leg

The leg is made of a bow-shaped spring and a string from tip to tip that hold it in compression. The spring leaf is fabricated from fiberglass with a small, plastic foot at the bottom end and an offset hip joint at the top. The fiberglass material is the type used in archery limbs (fiberglass bows), has a unidirectional fiber arrangement and volumetric glass content near 70% in an epoxy matrix. The material has a high specific energy capacity; our prototype legs typically store enough energy to lift the leg weight 100 meters or more in Earth's gravity. The current prototype is approximately constant in cross-section, and straight when unloaded. It takes a curved shape when the bow string is tensioned. The plastic foot provides a place to attach the bow string, as

[4] These are not really the Euler angles, I am actually using a small angle linearization.
[5] Except in § 5.1.3 where the stabilizing effect will be briefly analyzed.

well as an adjustment point for the control strings. It also gives a rounded surface for ground contact, and may be covered with rubber or other tractive material. The aluminum offset hip piece interfaces the fiberglass spring to the hip pitch bearing. The offset geometry reduces the stiffness at small deflections to soften impacts with the ground[6].

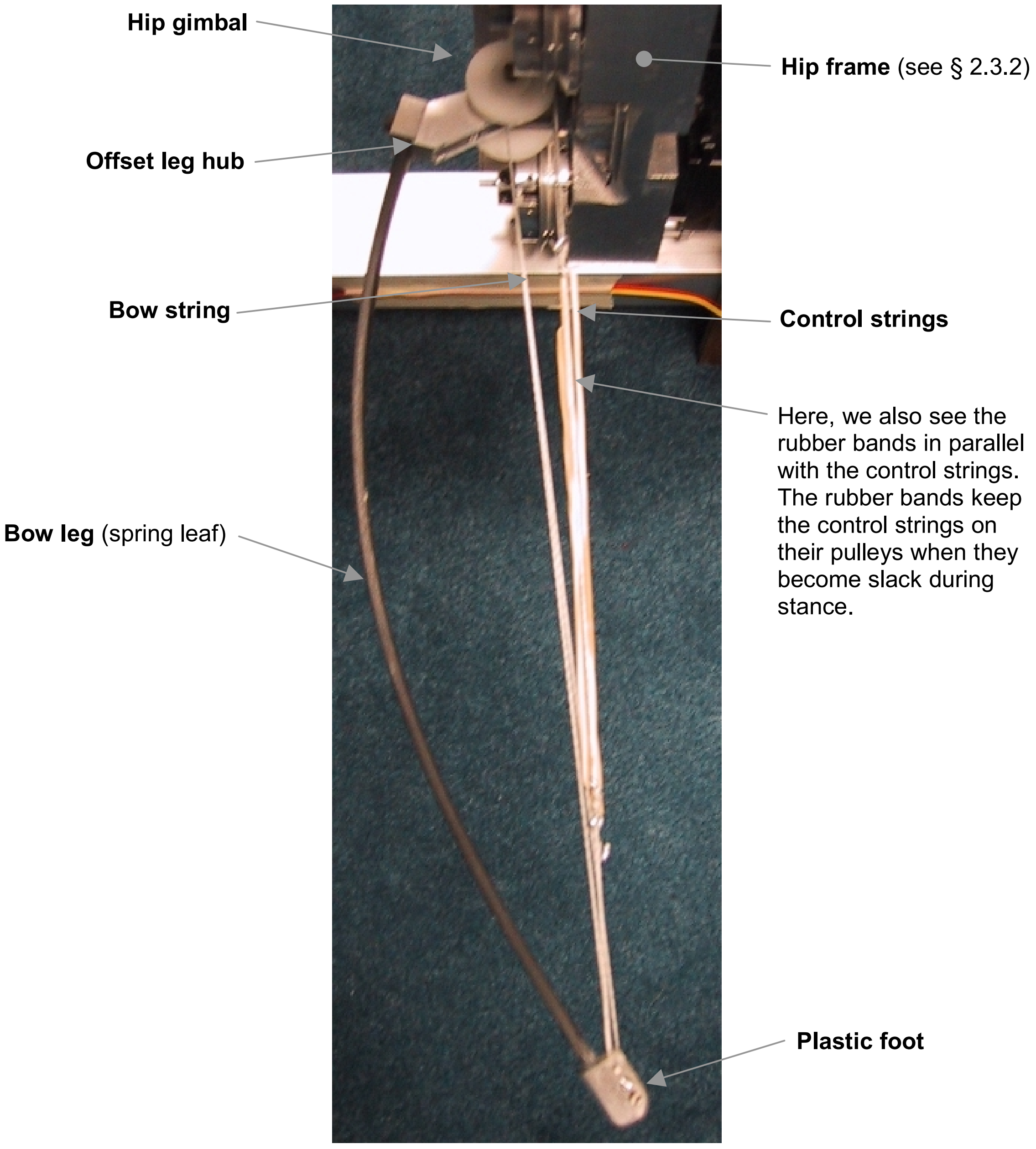

Hip gimbal
Hip frame (see § 2.3.2)
Offset leg hub
Bow string
Control strings
Here, we also see the rubber bands in parallel with the control strings. The rubber bands keep the control strings on their pulleys when they become slack during stance.
Bow leg (spring leaf)
Plastic foot

Fig. 2.7 – The bow leg

The force versus deflection characteristic is shown on Fig. 2.8. A more detailed description could be found in [Zeg99], p. 2.2. Especially, the Figure 2.1 correlates the energy stored in the leg with the area under the force/deflection curve.

[6] Explanations by Ben Brown, the inventor of the bow leg.

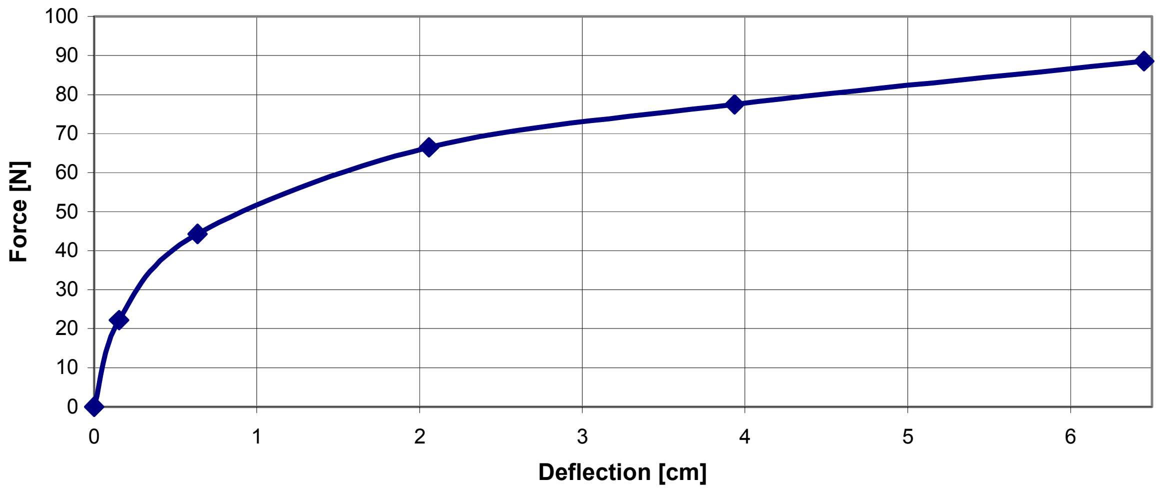


Fig. 2.8 – Force vs deflection graph for the employed bow leg

2.3.2 Gimbal hip and strings

The Fig. 2.9 shows the gimbal hip block. The hip frame holds the parts such as control servos, drive pulleys, which are driven by the servos, and the hip clevis, which is supported by a ball bearing.

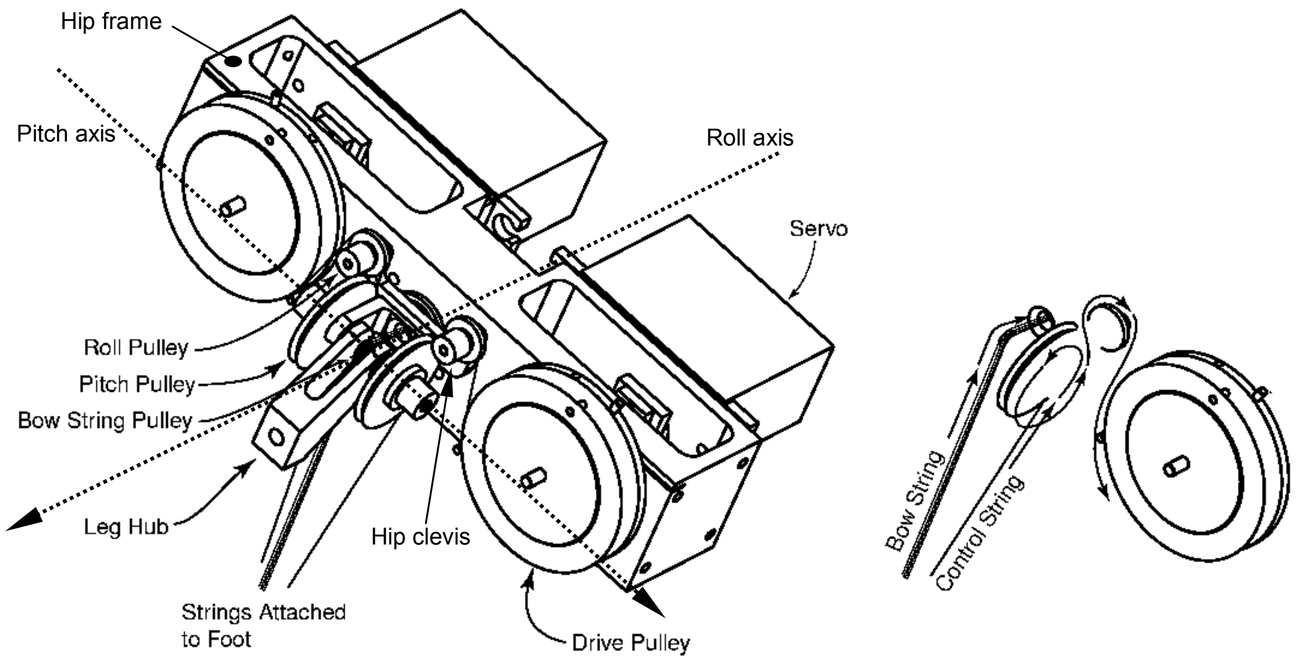


Fig. 2.9 (drawing by G. Zeglin) – Hip block schematic; control and bow string paths (right)

The hip clevis can thus rotate along the roll axis and serve as an attachment part for the roll pulleys, the pitch pulleys, the bow string pulley, and the leg hub. The pitch pulleys, the leg hub and the bow string pulley are mounted on the same axis that is co-incident with the pitch axis of the robot.

On Fig. 2.9, on the right, you can see the path of both the control strings and the bow string[7]. First the bow string links the foot to the thrust mechanism (see §2.3.4) by passing over the bow string pulley, which give it a little offset related to the pitch axis. This offset is positive, in the body frame *B* and thus tends pull the leg forward.

Then the control strings – one per control servo – follow a more complicated way. They are attached to the drive pulleys. Then, they first are supported by the little roll pulleys before making a whole turn around the pitch pulleys and going to the foot. Here the offset relative to the pitch axis is negative and longer than the one of the bow string because of the radius of the pitch pulleys (see also Fig. 2.10). This arrangement allows the control strings to compensate for the torque applied to the leg by the control string with a smaller tension, which is directly supported by the control servos. Therefore, activating the servos in a differential mode leads to a fore and aft movement of the leg (β_p).

The control strings have a symmetric offset in the pitch axis direction unlike the bow string. For this reason, a lateral displacement (β_r) of the leg may be obtained by commanding the control servos in a common mode.

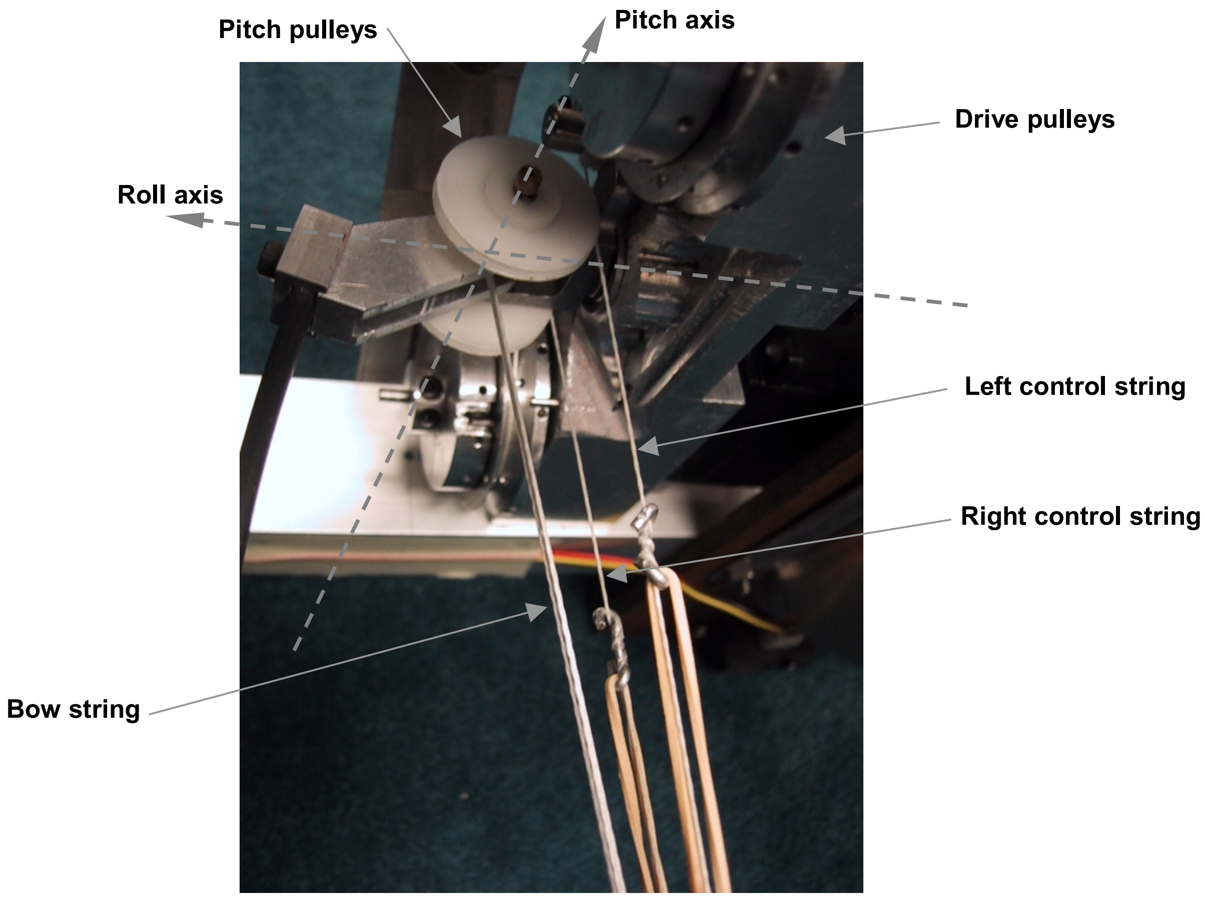


Fig. 2.10 – Bottom view of the hip block and the strings

When the control stings go slack during stance, they tend to go out of the pulleys. To alleviate this problem, a rubber band is attached in parallel to each control string (see Fig. 2.7) to maintain tension around the pulleys. Another springy element is present inside the drive pulleys. During stance, if one of the control string becomes tight because of the sweep of the leg, this spiral spring will allow the pulley to deflect and prevent high loads on the servos and minimize the torque disturbance on the body.

[7] the bow string is actually made of 4 strings in parallel

2.3.3 Body

The hopper body provides a structure for mounting the thrust mechanism, sensors, controller board and power supply. It has been constructed around the hip mechanism, which constitutes the core of the robot. Two aluminum channels have been attached to both sides of the hip frame as a support for transverse wood boards. Behind the hip frame is the thrust mechanism. At the opposite lies the controller. The battery packs[8] are mounted laterally in the aluminum channels.

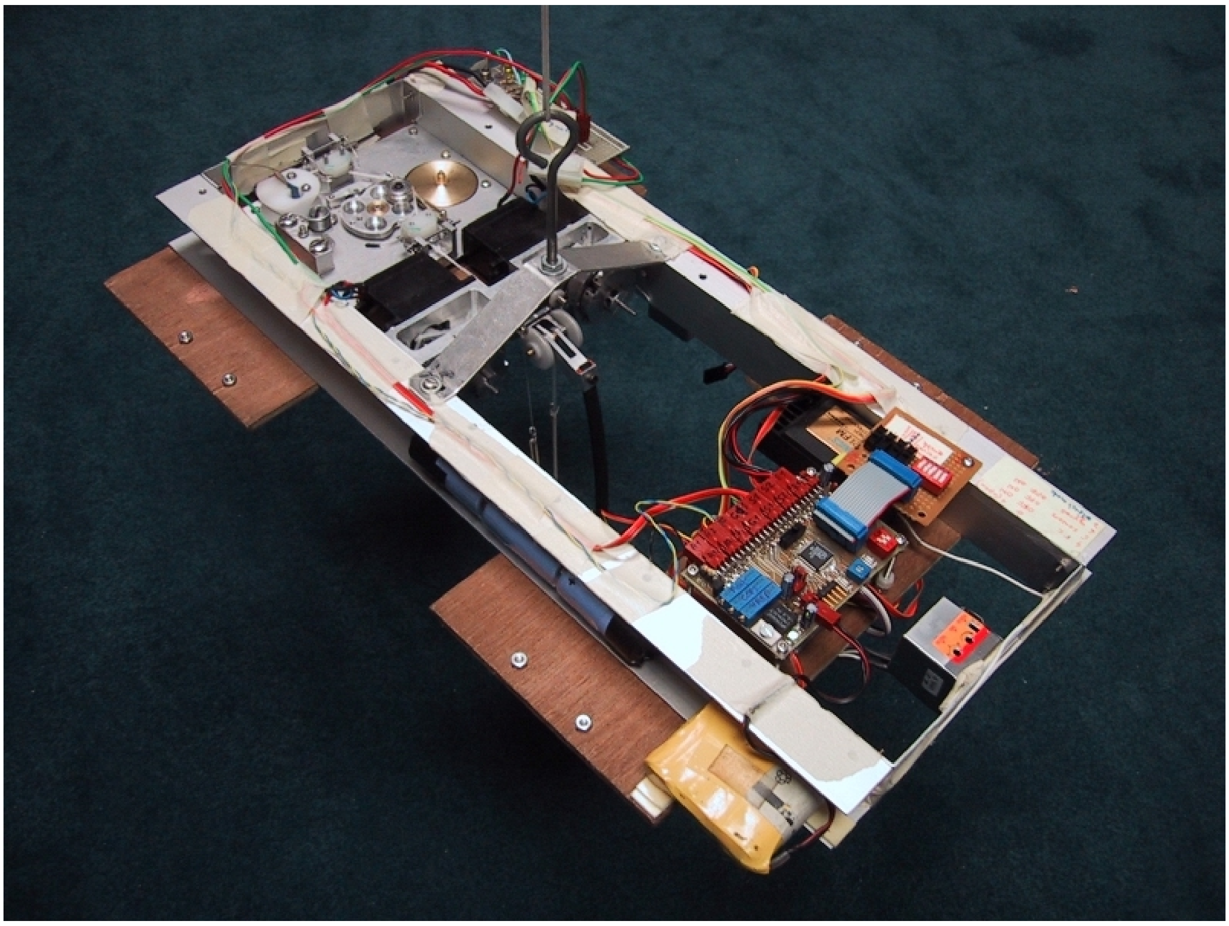


Fig. 2.11 – The body from left to right: thrust mechanism, hip frame and electronics

The body platform has to be well balanced: the center of mass (COM) must be situated exactly below the hip. In order to adjust the equilibrium, a tool has been built that allows the body to pivot precisely on the hip centerline about the pitch axis or the roll axis.

The weight of the robot is about 2kg, totally equipped. The body size is about 450x290x50mm and the leg length 230mm.

[8] One of five Sub-C cells for the actuators, another of four AA cells for the electronics and the sensors

2.3.4 Thrust mechanism[9]

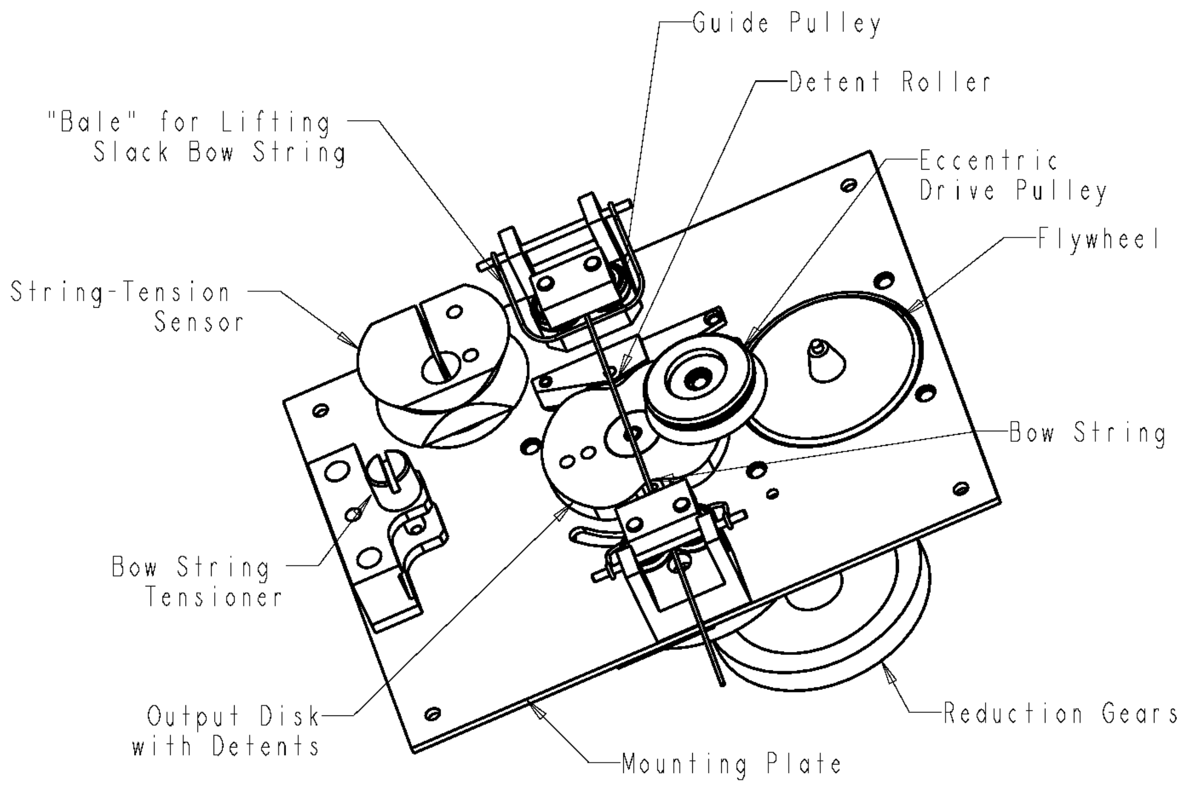


Fig. 2.12 (drawing by Ben Brown) – 3D view of the thrust mechanism

The thrust or leg-retract mechanism (Fig. 2.12) functions by means of an eccentric drive pulley that orbits and engages the bow string between two sets of guide pulleys. When the output disk rotates 180 degrees from the position shown in the figure, it drives the string laterally, increasing the length of the string between the pulleys, thereby retracting the leg. When the leg spring is compressed by ground contact, the bow string becomes slack and the two spring-loaded bales lift the string over the drive pulley, allowing the string to return to its straight configuration (Fig. 2.13). Thus, the leg extends to a greater length at liftoff, adding energy to the system. This cycle continues with the output disk carrying the drive pulley 180 degrees each hopping cycle, injecting a fixed quantum of energy each cycle.

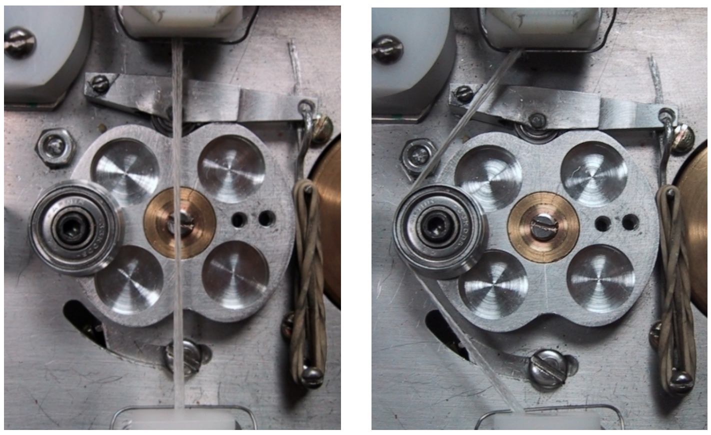


Fig. 2.13 – Real thrust mechanism: relaxed (left) and cocked (right)

[9] Described by its designer: Ben Brown.

The original retract mechanism on the 3D Hopper, like that on the 2D Hopper, used a hobby servo to drive the output disk back and forth through a 180 degree reciprocating motion. However, such servos are typically too slow and/or weak to inject adequate energy quanta at a high enough rate[10] for hopping in full gravity (see experiment under § 6.4). Further, these actuators operated at high load and relatively low speed, so motor efficiency was low and power consumption high. A search for commercially available gearmotors uncovered nothing of suitable torque, speed and weight. A new concept was needed to permit full-gravity operation.

The present version of the retract mechanism (Fig. 2.13) on the 3D Hopper uses a flywheel continuously driven by a motor operating near its optimally efficient speed. The motor/flywheel is connected to a drive shaft through a 2-stage gear reducer with 20.5:1 ratio. A wrapped-spring clutch, activated by a small electromagnet, intermittently couples the drive shaft to the output disk, driving it forward. A pair of stop pins on the clutch, and detents on the output disk, cause the disk to index precisely 180 degrees each activation cycle. A force-sensing resistor (FSR) sandwiched in the gap of the string tension sensor, provides a signal used to indicate stance and flight phases of the hopper. The bow string[11] wraps around the string tension sensor such that the string tension squeezes and loads the FSR. The bow string tensioner allows adjustment of bow string length/tension.

2.4 Sensors

In this section all the used sensors will be briefly presented in order to give an overview of the current perception of the robot. The four range finders, which have been mounted below the body in order to compute its attitude, will be discussed in details in Chapter 4.

➢ **Gyros**

As explained in § 4.5, a quite important delay[12] exists in the body attitude computation. In order to have a mean to alleviate this problem, the body is equipped with two hobby gyros[13]. One of them provides information about the roll angle velocity and the second about the pitch angle velocity.

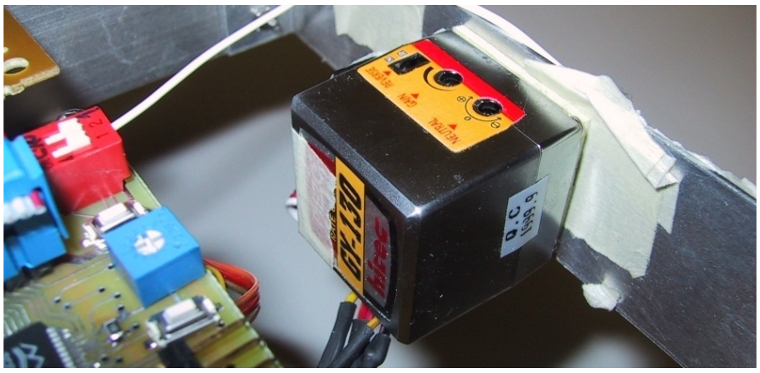


Fig. 2.14 – One of the two gyros that equip the body

➢ **Foot sensor**

Another important information to have is the time of ground contact. A force sensing resistor has been placed along the path of the bow string to sense string tension, between the retract mechanism and the tensioner (see § 2.3.4). This variable resistor is part of a resistor divider, whose output voltage is connected to a comparator. This system provides binary information about the state of the machine: `FLIGHT` or `STANCE`.

[10] Typically 2-4Hz.
[11] Not fully shown in the figure.
[12] About 100ms.
[13] Piezo gyro HITEC GY-130: weight 26.6gr, size 28.5 x 28 x 29.4mm, voltage 4.8 to 6V.

3 Software & electronics

3.1 Microprocessor system

3.1.1 Selection of the microcontroller

A microcontroller is needed for hosting the controller program and managing all the low level tasks. As described in the introduction, the first controller that has to be implemented is not very complex: it does not take care of the global lateral velocity of the robot or the path planning and foot placement.

The low level tasks are:

- Reading the sensors at a high rate.
- Generating the actuator signals such as PWM for the servomotor.

Moreover the selected processor should be able to communicate with a PC for debugging and analysis purpose (see § 3.3). In order to simplify the development an *in-system programmable* capability is required.

Several microcontrollers as well as existing microcontroller boards have been evaluated. Very often these boards are quite complex and not well adapted for our purposes: too large or for all-purpose. A lot of processors are too powerful (e.g. Motorola 68'xxx) for the task or a little too slow and limited (e.g. Microchip PIC). Finally, we chose the Ubicom[14] SX52BD100 as a good compromise between speed and complexity. This selection is also the result of preliminary encouraging trials with the SX28AC that turns out to have not enough memory.

3.1.2 Ubicom SX52BD

The selected microcontroller is presented in some details in this paragraph. In the rest of the text, it will be referred simply as the 'microcontroller'.

The Ubicom SX52BD100 has a Harvard RISC-based architecture that allows high-speed computation, flexible I/O control and efficient data manipulation. The very high operating frequencies (up to 100MHz)[15] and the mostly single cycle instructions enhanced the throughput. In addition, the architecture is totally reprogramable and deterministic thus enabling the device to implement hard real-time functions as software modules.

Below the most significant features related to our application are listed. For more detailed information, please consult the data sheet available at [wwwUbicom].

- **Key features**
  - Speed: 20ns instruction cycle, 60ns internal interrupt response at 50MHz
  - Program memory: 4096 words (12-bit) of EE/Flash (more than 10000 rewrite cycles).
  - Data memory: 262 bytes of SRAM distributed into 16 banks.
  - Compact instruction set, which does not contain DIV or MUL.
  - Fast table lookup capability through run-time readable code (IREAD instruction).
  - Fast and deterministic (jitter-free) interrupt: hardware context save and restore of key resources such as PC, W, STATUS and FSR within the 3-cycle interrupt response time.

[14] Formerly Scenix.
[15] In our case it will be operated at 50Mhz because first it is fast enough, second some pieces of software where already written for this frequency and third it limits the power consumption.

- Flexible I/O: all pins are individually programmable as input or output, TTL or CMOS compatible, all pins have selectable pull-ups. Some pins have special features like Schmitt Trigger or analog comparator.

➢ **Programming and debugging**

- On-chip in-system programming via oscillator pins.
- On-chip in-system debugging support logic.

➢ **The Virtual Peripheral concept**

A Virtual Peripheral (VP) is a software module that replaces a traditional hardware peripheral in taking advantage of the SX high speed and deterministic nature to produce same results as the hardware peripheral with much greater flexibility. For instance, this concept has been used to implement RS232 serial communication, SPI interface for AD converters and so on.

3.1.3 Modules

In this section, I will briefly present the different modules[16] that have been implemented in the microcontroller. That shall also allow me to formulate an overview of my program architecture.

Only one source of interruption is enabled which is the internal RTCC overflow. Therefore, the program is essentially organized into an interrupt routine that is executed every 4.34μs[17] and a main function that encloses a perpetual loop starting every millisecond.

➢ **Real time counter**

The real time counter is a simple timer that increments each millisecond, based on the interrupt rate (4.34μs). It represents the real-time base of our application, allowing the synchronization of the main loop. Thus all the operations in the main routine are repeated every millisecond (such as sensors reading) or a multiple of 1ms (like the body attitude computation process described in § 4.5, which recurs every 40ms).

➢ **AD converter (SPI)**

An SPI routine has been implemented that interfaces an 8-channel AD converter for sensors reading. This function is not executed in the interrupt routine unlike the UART interface described below. This is a quite simple piece of code that sends a control byte to the converter, waits for the conversion and reads back the result, using the well-known SPI serial protocol and providing the clock. Note that the microcontroller is a little bit too fast for the converter SPI, thus a few NOP commands must be inserted.

➢ **UART interface (RS232)**

This one has been implemented as a VP. That means it is executed in the interrupt routine. This functionality has been adapted from a library provided with the compiler (see § 3.2) in order to reach the rate of 115200bps. The speed is critical in this case, because this routine will provide great debugging capability. So the faster it is, the larger set of values it will be able to send to the host PC during the unused time.

Note that, for the moment, this interface is only one way because it is not very useful to send data to the microcontroller. As we shall see in § 3.3, this UART is mostly for supervising purpose.

[16] Some of them are Virtual Peripherals.

[17] The `RTCC_PER_INT` is set to 217, which leads to an interrupt every 217 clock cycles. As the clock is set to 50MHz, one clock cycle last 20ns.

- **Radio command**

This routine has been written by Matthias Felber in assembler and reused without important changes in my program. It implements the capability of reading three channels from a standard hobby radio receiver. This will allow adding a user command on top of the controller output.

- **Servomotors**

The standard hobby servomotors[18] require a PWM signal. The pulse duration determines the commanded position of the servo. The routine that implements this functionality for the two control servos takes advantage of the interrupt routine. The pulse starts are ordered by the main function every 10ms (update rate of 100Hz)[19] and the interrupt routine takes care of their end with a precision of 4.34μs.

The same principle is also used to generate a 10ms pulse for the solenoid that engages the clutch for the thrust mechanism.

- **Gyroscopes**

Two hobby gyroscopes are employed. A PWM signal similar to those sent to the servos must be provided. The pulse duration may be constant. The gyros correct this signal by modifying the duration of the pulses, which are read back by the microcontroller. In the interrupt routine, there is a counter that is incremented every time the corresponding pin is at a high level and reset after a transition. Another variable is updated with the duration of the pulse and can be read by the main function.

The driving PWM signal has a frequency of 200Hz what determines the update rate of the gyros information.

- **LCD**

This module has been adapted from a library provided with the compiler. It provides a set of functions for handling a standard 8x2 characters display with a parallel interface.

This debugging tool is not used very often because it is not so fast and the UART provides a much more complete and flexible overview (see § 0). However, it could be useful for quick debugging, during unconnected experiments.

3.1.4 Electronic board and components

Based on the preliminary experience with a hand-wired microcontroller board, Garth Zeglin has designed a new PCB exactly adapted to our application needs. For instance, reliable interconnections are critical for this hopping robot. A large number of sensors and actuators must be connected to the microcontroller (see Fig. 3.1) and the vibrations must not disturb the signals. For more convenience, each sensor or actuator has its own connector.

[18] The employed servos, Airtronics 94257, have the best possible ratings among the available hobby servos.
[19] At the beginning, I tried with PWM rate of 200Hz but the servos sometimes entered in a strange mode with oscillation around another position than the one required.

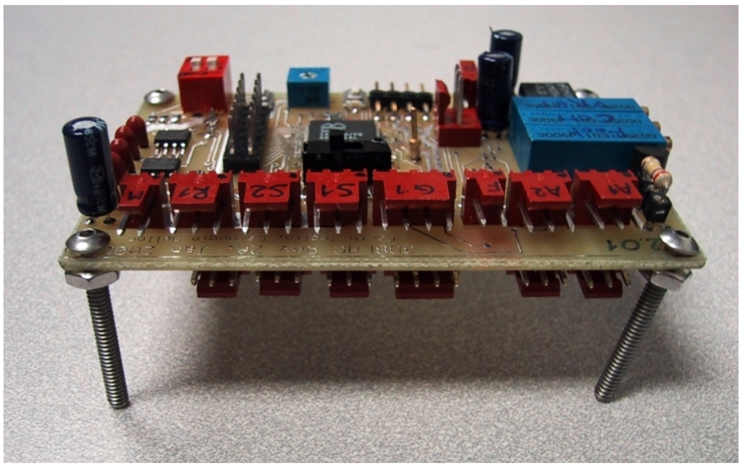


Fig. 3.1 – The microcontroller board: specifically designed for the 3D Bow Leg Hopper

The schematics of this board can be found in Appendix 8.5.

Here below are listed the most important chips of this board:

➢ **AD converter**

Chip name: MAX148BCAP (Maxim)

Package: 20 pins SSOP

Key features:

- 8-channel single-ended or 4 channel differential inputs.
- Single-supply operation from +2.7V to +5.25V.
- SPI-compatible 4-wire serial interface.
- Speed: sample to 133ksps.

➢ **RS232 driver-receiver**

Chip name: MAX232ACSE (Maxim)

Package: 16 pins Narrow SO

Key features:

- Operate from Single +5V Power Supply

➢ **External memory**

This memory is not in use for the moment but the board has spares for two memory chips in case it would become necessary to enhance the memory capability of the SX52BD.

Chip name: 25LC640 (Microchip)

Package: 8 pins TSSOP

Key features:

- Memory: 64K bit organized as 8192 x 8 bit in 32-byte pages
- Serial protocol: SPI
- Write cycle: 5ms maximum.
- Self-timed ERASE and WRITE cycles

3.2 Development environment

From the beginning, in order to reduce the development time, the decision was made to develop the program mainly in C. For this purpose, we have purchased the SXC professional compiler from ByteCraft [wwwByteCraft], version 2.0.

This quite expensive tool is incredibly powerful but before briefly presenting its qualities, I must say that, for the moment, it contains a few unpleasant and time-consuming bugs[20].

It is a real ANSI C-compiler adapted to the SX microcontroller. The resulting assembler is directly accessible within the user interface and correlated to the C program such that it is very easy to analyze the output code. The resulting assembler is well optimizing by several processes during compilation. The SXC has also a built-in linker that allows for multiple files projects.

The SXC is released with a set of useful libraries like the C-standard ‘stdlib.h’ or ‘stdio.h’ and some device-specific headers like ‘sx52bd.h’, ‘startup.h’, etc. It has also some integrated routines to compensate for the lack of division or multiplication in the basic operations of the SX microcontroller. It can even manage floating-point operations, if we don’t worry about the consumed time.

One of the greatest advantages is that the user does not have to worry about bank switching in data memory and page switching in program memory.

Parts of the program can be written directly in assembler. But care must be taken because doing so tends to break the capability of the C-compiler to accurately handle the bank and page features of the microcontroller.

3.3 Supervising the robot

As seen in § 3.1.3, a RS232 virtual peripheral has been implemented on the microcontroller, which is therefore able to send data to the PC at 115’200bps.

In this section, the BLHSupervisor will be presented. This is the program, which receives the data from the microcontroller and interprets them. I have developed this tool for several purposes:

1. Displaying interesting values in real time for debugging.
2. Recording data for analysis and debugging.
3. Lookup table building.

For instance, it allows building the characteristics shown in chapter 4. These values can also be employed to construct the lookup table (see § 4.2.3). Actually, the BLHSupervisor is also able to prepare the lookup table in assembler language that could be directly inserted into the program for the Scenix.

The program is a multi-threaded[21] application written in C++ with Borland C++ Builder 4. The last revision of the software is very flexible and capable of operating in different modes:

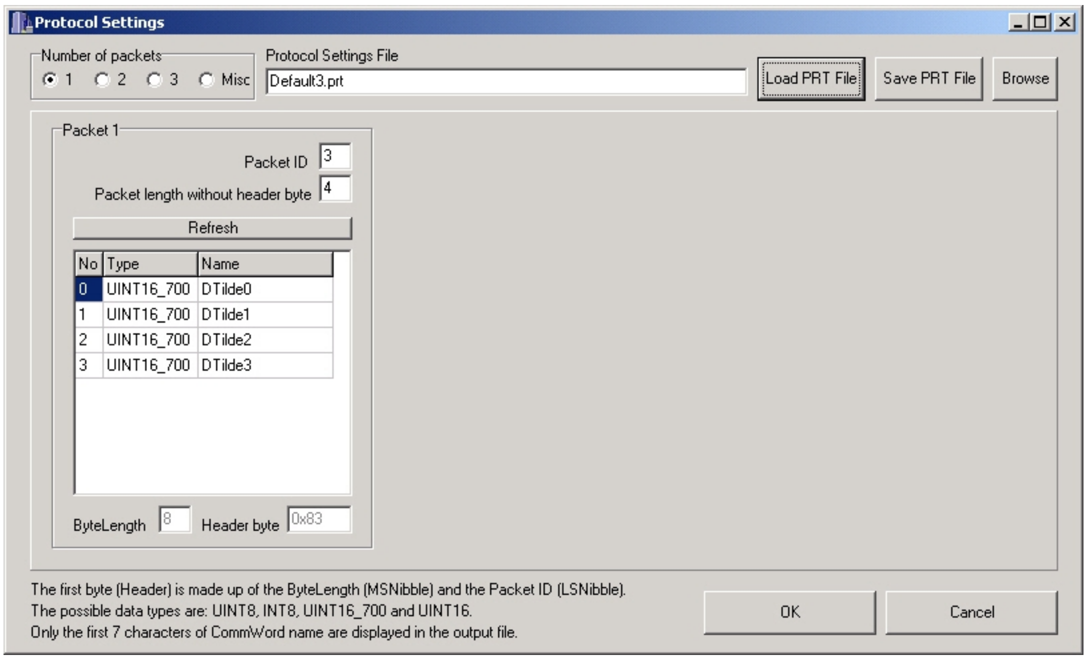
➢ **Single packet**

A packet is a set of values that will be repetetively sent by the microcontroller, for instance each 40ms. Each value in the packet can be defined as `UINT8` (unsigned integer on 8-bit), `INT8` (signed integer on 8-bit), `UINT16_700` (unsigned integer on 16-bit within the range from 0 to

[20] Here is a non-exhaustive list: arrays handling, conversion functions in stdlib, problem with the list file size when sending to the programmer, lack of in-system debugging functionality.

[21] One thread handles the continuous communication flux and another one, the main thread, is essentially responsible for the user interface. A shared memory configuration is used.

700) or `UINT16` (unsigned integer on 16-bit). A header byte is used to identify the packet (on the low nibble) and determine its size (on the high nibble). The size[22] of a packet can thus vary between 1 to 15 values. Fig. 3.2 shows the dialog box that allows to set the parameters.



Protocol Settings

Number of packets: [✓ 1] 2 3 Misc

Protocol Settings File: Default3.prt

Packet 1

Packet ID: 3

Packet length without header byte: 4

No	Type	Name
0	UINT16_700	DTilde0
1	UINT16_700	DTilde1
2	UINT16_700	DTilde2
3	UINT16_700	DTilde3

ByteLength: 8 Header byte: 0x83

The first byte (Header) is made up of the ByteLength (MSNibble) and the Packet ID (LSNibble).
The possible data types are: UINT8, INT8, UINT16_700 and UINT16.
Only the first 7 characters of CommWord name are displayed in the output file.

Fig. 3.2 – Single packet protocol settings

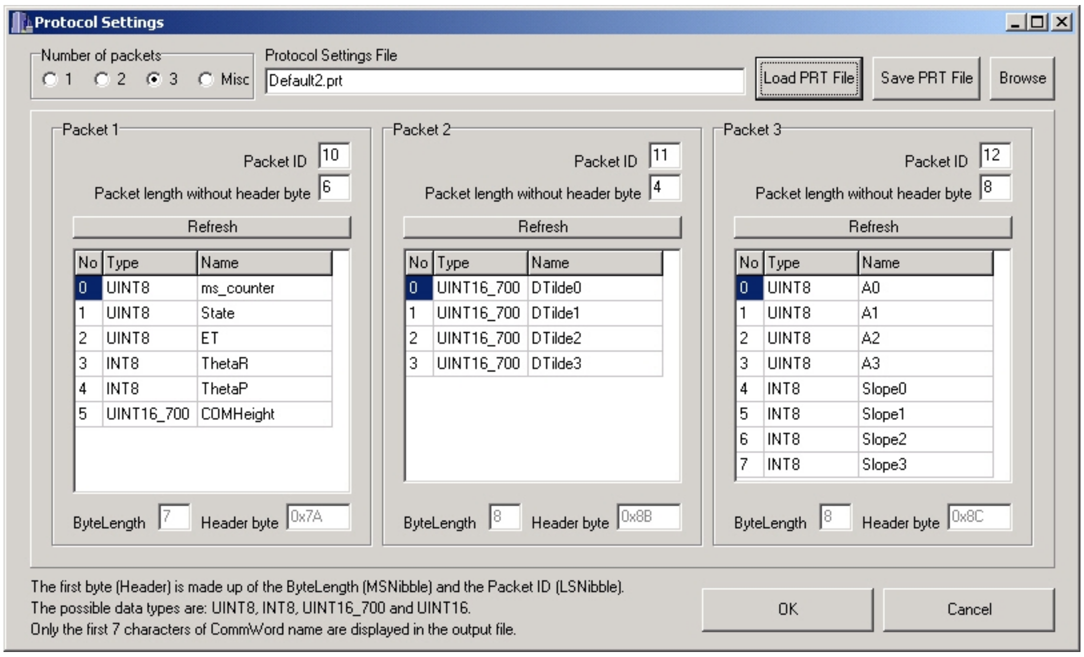
This interface allows the user to define the packet and computes automatically the header byte that must be copied into the controller program. For example, the Fig. 3.2 shows a packet made of 4 unsigned integer values on 16-bit. These values are limited to 700 because they represent the distances measured by the sensors. The header byte is 0x83 because the length of the packet in bytes is 8 and the packet identifier is 3.

Actually, the time remaining in the base loop of the main program – executed every millisecond - permits to send about 8 bytes, at the moment. But quite often, an update rate of 40ms for these values is enough (see § 5.3.3). If we need to transmit more than 8 bytes, the multiple packets mode may be employed.

➢ Multiple packets

This mode allows for sending multiple packets one after another with a delay of at least1ms between each. Each packet is self-contained and must have its own identifier. Whit this mode, the controller must send more important set of interesting data that are updated at 40ms, for instance. Fig. 3.3 shows a protocol with a bunch of values like all the distances, the analog values from 4 AD converter channels and so on.

[22] The size - `ByteLength` - of the packet is here the number of bytes, which is different of the number of values that are sent - `Length` - because those values could be made of two bytes, for instance `UINT16`.



Number of packets: 1 2 [✓ 3] Misc

Protocol Settings File: Default2.prt

Packet 1 — Packet ID: 10 — Packet length without header byte: 6

No	Type	Name
0	UINT8	ms_counter
1	UINT8	State
2	UINT8	ET
3	INT8	ThetaR
4	INT8	ThetaP
5	UINT16_700	COMHeight

ByteLength: 7 — Header byte: 0x7A

Packet 2 — Packet ID: 11 — Packet length without header byte: 4

No	Type	Name
0	UINT16_700	DTilde0
1	UINT16_700	DTilde1
2	UINT16_700	DTilde2
3	UINT16_700	DTilde3

ByteLength: 8 — Header byte: 0x8B

Packet 3 — Packet ID: 12 — Packet length without header byte: 8

No	Type	Name
0	UINT8	A0
1	UINT8	A1
2	UINT8	A2
3	UINT8	A3
4	INT8	Slope0
5	INT8	Slope1
6	INT8	Slope2
7	INT8	Slope3

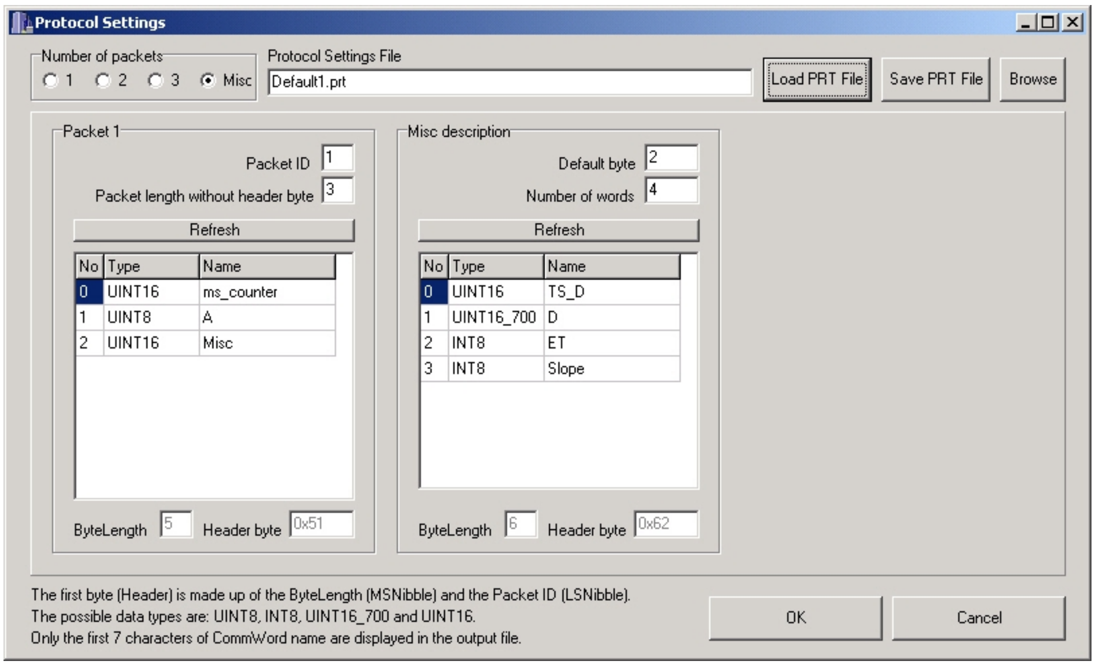
ByteLength: 8 — Header byte: 0x8C

The first byte (Header) is made up of the ByteLength (MSNibble) and the Packet ID (LSNibble).
The possible data types are: UINT8, INT8, UINT16_700 and UINT16.
Only the first 7 characters of CommWord name are displayed in the output file.

Fig. 3.3 - Multiple packets protocol settings

➢ **Single packet with a miscellaneous word at its end**

In order the debug the routines, which output updated data every millisecond and some other important values in an asynchronous manner[23], this mode can be used.



Number of packets: 1 2 3 [✓ Misc]

Protocol Settings File: Default1.prt

Packet 1 — Packet ID: 1 — Packet length without header byte: 3

No	Type	Name
0	UINT16	ms_counter
1	UINT8	A
2	UINT16	Misc

ByteLength: 5 — Header byte: 0x51

Misc description — Default byte: 2 — Number of words: 4

No	Type	Name
0	UINT16	TS_D
1	UINT16_700	D
2	INT8	ET
3	INT8	Slope

ByteLength: 6 — Header byte: 0x62

The first byte (Header) is made up of the ByteLength (MSNibble) and the Packet ID (LSNibble).
The possible data types are: UINT8, INT8, UINT16_700 and UINT16.
Only the first 7 characters of CommWord name are displayed in the output file.

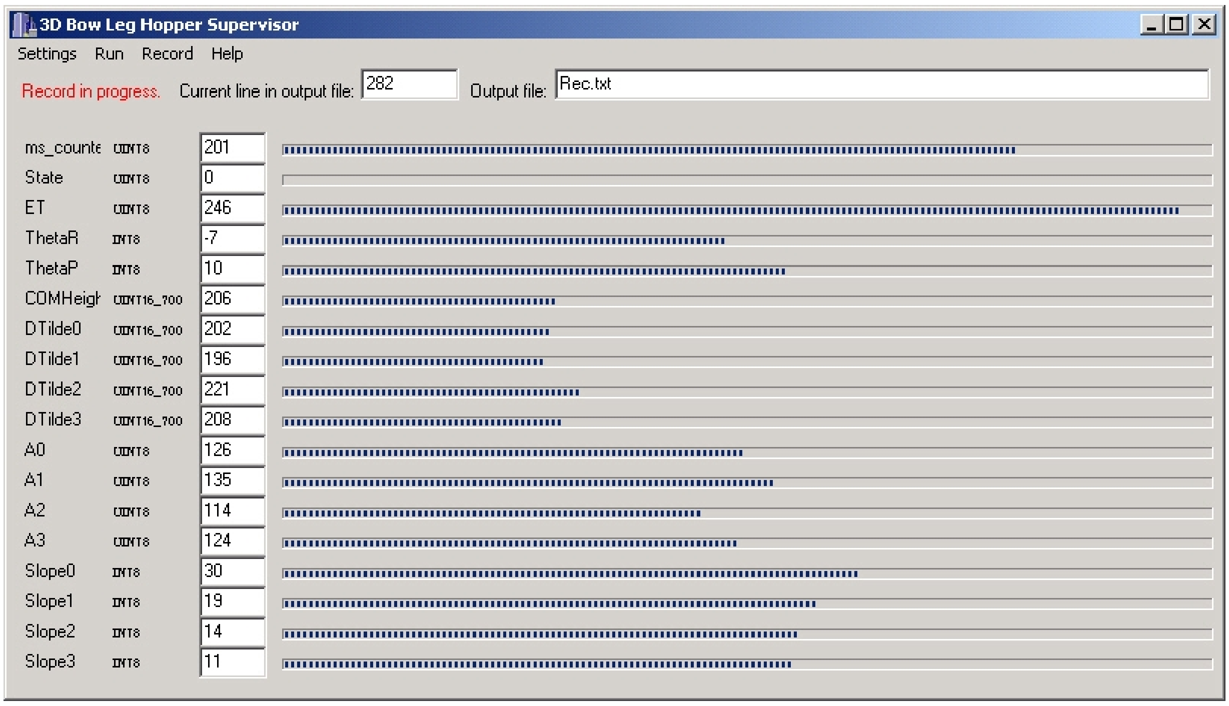
Fig. 3.4 - Single packet with a miscellaneous word at its end

The second packet called «misc description» determines the interpretation of the two last bytes of the first packet. When the header of the second packet is detected in the last word of the first

[23] Typically, the analog values from the sensors are sent every millisecond as well as their time stamp. Then, when a distance is computed, asynchronously, some significant values can be sent. A good example is the graph of the Fig. 4.22.

one, a counter is set to the value of the *misc packet* length. Then, when the next first packets are received, they must contain, in their last byte, the value of the *misc packet*, one word after another.

Finally, Fig. 3.5 shows the result of one of these settings displayed in real time on the main interface. The record capability allows to log the successive data into a text file.



Record in progress. Current line in output file: 282 Output file: Rec.txt

ms_counte	UINT8	201
State	UINT8	0
ET	UINT8	246
ThetaR	INT8	-7
ThetaP	INT8	10
COMHeigh	UINT16_700	206
DTilde0	UINT16_700	202
DTilde1	UINT16_700	196
DTilde2	UINT16_700	221
DTilde3	UINT16_700	208
A0	UINT8	126
A1	UINT8	135
A2	UINT8	114
A3	UINT8	124
Slope0	INT8	30
Slope1	INT8	19
Slope2	INT8	14
Slope3	INT8	11

Fig. 3.5 – BLHSupervisor main interface

The other possibilities of this software – like the building of lookup tables or the display of a rough 3D drawing of the Hopper moving in real time – are not presented here.

This tool was and will be of primary importance during debugging complex routines implemented in the microcontroller.

4 Sensing the body attitude

This chapter discusses sensing of body attitude, which was done on the planar hopper by a potentiometer mounted between the boom and the body. One way to accomplish this task on the unconstrained machine - without involving vision or offboard sensors - is to measure several distances between the body and the ground. For this purpose, I had to select appropriate distance sensors.

The function of the distance sensors is to deliver the height and the attitude of the body in real time. Therefore, at least three sensors must be placed under the body, oriented toward the floor: three values are needed to compute the three parameters: θ_r, θ_p and h.

For several reasons, I chose to mount four sensors. The first is that the computation of roll (θ_r) and pitch (θ_p) angles can be totally uncorrelated. Then it is easier to place them evenly around the hip. And with four sensors, there is recurrent information (for instance to compute the height of the COM) that allows averaging several results for improving the noisy information.

This chapter is organized as a progression from sensor selection (§ 4.1) to dynamic computation of the body attitude. Because there are some important complications when the sensors are used dynamically, I will first discuss their static comportment (§ 4.2) and their geometry on the body of the robot (§ 4.3) before describing a way to alleviate the dynamic measurement problem (§ 4.4). Finally, I will explain the global procedure, which allows computing the body attitude in real time (§ 4.5).

4.1 Selection of the distance sensors

The most important criteria for such sensors are:

1. Range of measurable distances
2. Update period (for dynamic measurements)
3. Synchronization between several sensors
4. Direction of the measured distance
5. Maximum admissible angle of the obstacle
6. Disturbance between several sensors reading the distance in the same direction

Further features such as precision, type of output, power requirement, weight, etc. are also important, but the above criteria are enough to make a first choice.

We consider two types of range finders that are appropriate for this application: ultrasonic and laser triangulation. The IR sensors based on the reflected light amplitude are not precise enough and too dependent on the surface color and type. Other range finders are either too expensive or have not the correct range.

Because they are not too expensive and already available, I evaluated two sensors, one of each aforementioned type: the Sharp GP2D12 [wwwSharp] and the MiniA [wwwSonaSwitch].

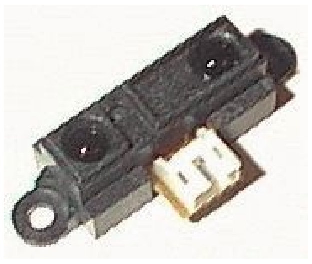
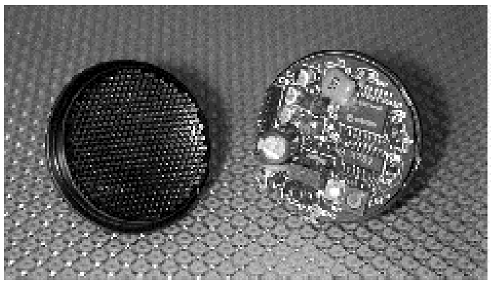



Fig. 4.1 – The evaluated sensors: the Sharp GP2D12 on the left and the SonaSwitch MiniA

Here is a brief comparative table:

	Sharp GP2D12 (IR triangulation)	**MiniA** (ultrasonic)
Output type	Analog, non linear	Analog, linear
Range	10 to 80cm	15 to 300cm
Update period	~40ms	> 20ms (with ext. trigger)
Synchronization between several units	No	Yes
Direction of the measured distance	Very directive, in the direction of the IR LED	Not well defined, about the nearest distance to the obstacle
Max admissible angle on flat surface	> 40°	> 22° at 600mm
Power supply voltage	4.5 to 5.5V	8 to 16V
Noise on the analog output	< 200mV	~300mV

Table 4.1 – Sharp GP2D12 - MiniA comparison

The MiniA is interesting because of the possibility to externally trigger it such that *simultaneous* distances can be obtained from several sensors. If they are activated together, there is not an important risk of disturbance between several sensors because the way of the sound from one sensor to another is always longer than the direct way back to itself. The main drawback of this sensor is the limited admissible angle on flat surface. Other disadvantages are the weight, the bulk and the supply voltage different from the standard 5V used for our onboard electronics.

For these reasons I chose the Sharp GP2D12. Unfortunately, it has also an important drawback: it has no external trigger such that it is impossible to synchronize several sensors. This important issue will be discuss later on in more details (see § 4.4 & 4.5).

The maximum consumption of this sensor is 35mA. The LED is certainly internally modulated such that a lot of noise is present on the power supply lines. A good condenser can help to reduce it. Another characteristic of this range finder is the very well defined direction of the measured distance due to the collimated infrared LED.

Finally, the Sharp sensors are very cheap and lightweight, but are not very fast and not synchronizable. There are other similar sensors that are much more dynamic. However, they are heavy and very expensive. A comparison between several range finders may be consulted on my web site [wwwjczSensors].

4.2 Static distance measurement

4.2.1 AD conversion of the sensor analog output

When powered with a 5V potential, the Sharp sensors have a maximum output voltage of about 2.45V, for close distances. The highest useable distance gives approximately 0.45V.

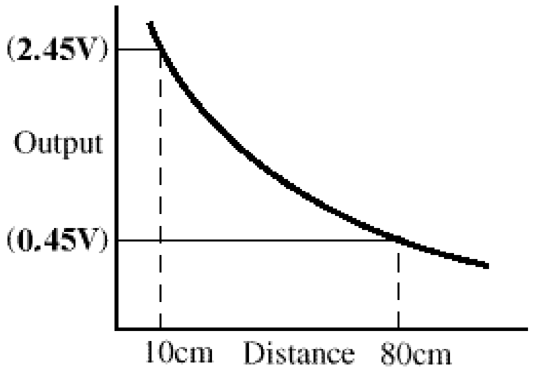


Fig. 4.2 – Output pattern of the Sharp GP2D12

Two external adjustable voltage references are used (Vref and COM). The highest limit of the converter, 255 on 8 bits, corresponds to the highest output voltage of the sensor, that is the lowest measurable distance, and vice versa.

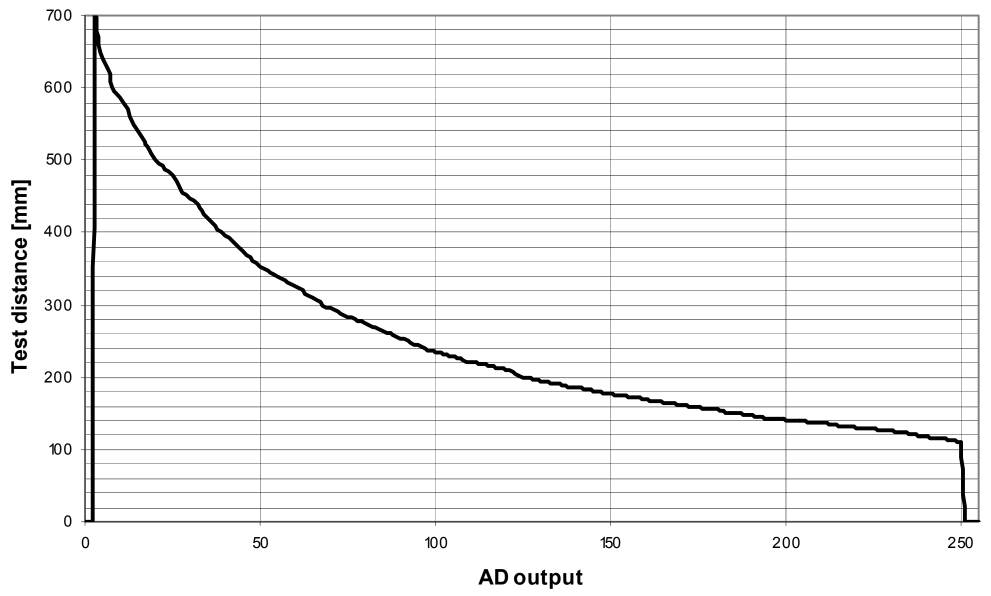


Fig. 4.3 – Measured characteristics of the Sharp GP2D12

So the full scale of the converter is used which improves the precision when employing an 8-bit lookup table (see §4.2.3).

4.2.2 Statistical analysis of the sensor output

In order to have an idea of the noise present on the sensor output, I made some tests and statistics with a Sharp sensor placed at about 25cm in front of a flat white wall. The distribution of 10'000 successive values acquired by the microcontroller (at 1kHz) is shown on Fig. 4.4.

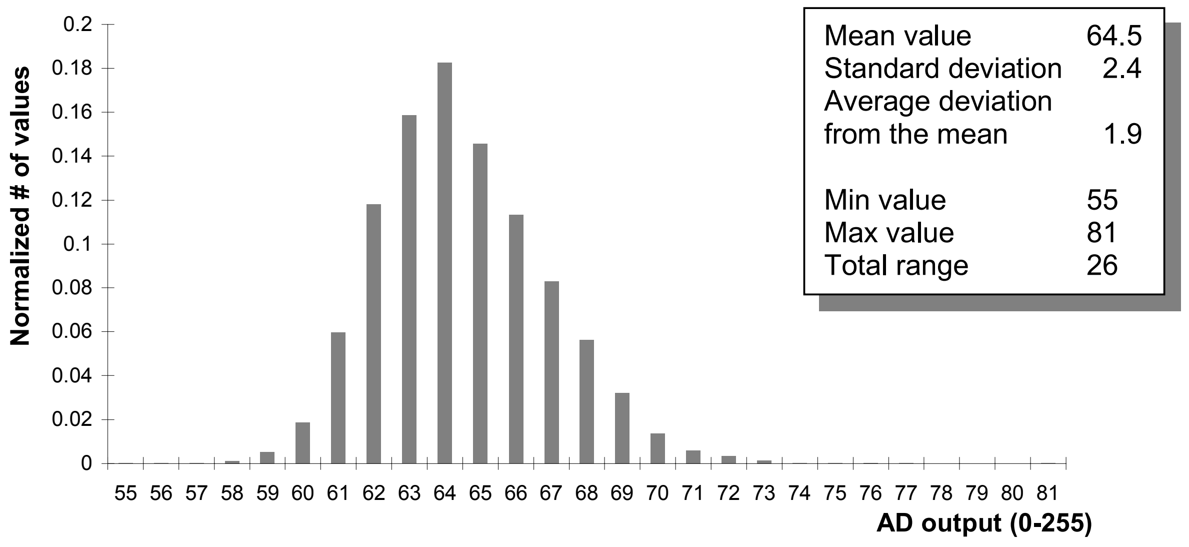


Fig. 4.4 – Statistical analysis of the Sharp GP2D12 sensors output

The values presented here are characteristic of the Sharp sensor output. In particular, the standard deviation does not depend on the measured distance. However, the precision of the distance cannot be directly deduced because the output is non linear. A constant standard deviation over the full range of the sensor output does not lead to a constant precision of the computed distance. See § 4.2.3 for more information about this.

This graph shows that the noise of the Sharp sensor output follows a normal-like distribution law. It would be a good idea to average several successive output values to improve the reliability.

Note that this quite important noise is essentially due to the sensor itself. The noise introduced by the converter is marginal. See Fig. 4.17 to convince yourself.

4.2.3 Linearization and lookup table

As the employed 8-bit micro controller is not able to efficiently handle multiplications and divisions, a lookup table must be used to convert the output values of the sensors into distances. As the converted values are 8-bit coded, the best we can do is to use a lookup table with 256 entries. The microcontroller offers the possibility to store 12-bit data in the program memory. It is therefore easy to store the distances in millimeters (from 100 to 800) that would require 10 bits.

As the microcontroller has not a lot of program memory, it would seem wise to have a single table for all the four sensors.

In order to build this lookup table, the BLHSupervisor (see § 3.3) has been employed. It acquires 400 successive values for each sensor and for each distance and then computes an average over time of the values.

The four sensors are arranged on a line, as shown on Fig. 4.5, such that they all *see* the same distance to the wall. A set of 40 reference distances has been drawn on the desktop. Lines parallel to the wall (which acts as the floor for these measurements) represent these distances in order to ensure good alignment. Due to the non-linear output of this sensor, the chosen reference distances are distributed in a non-linear manner, too.

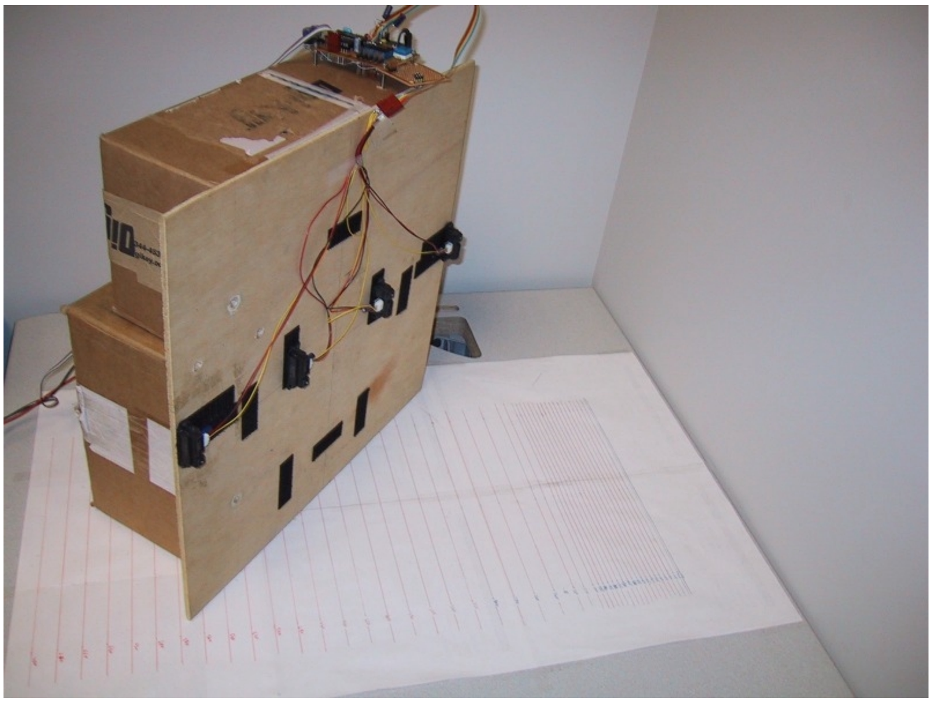


Fig. 4.5 – Setup for building lookup tables

The result of this process is plotted here:

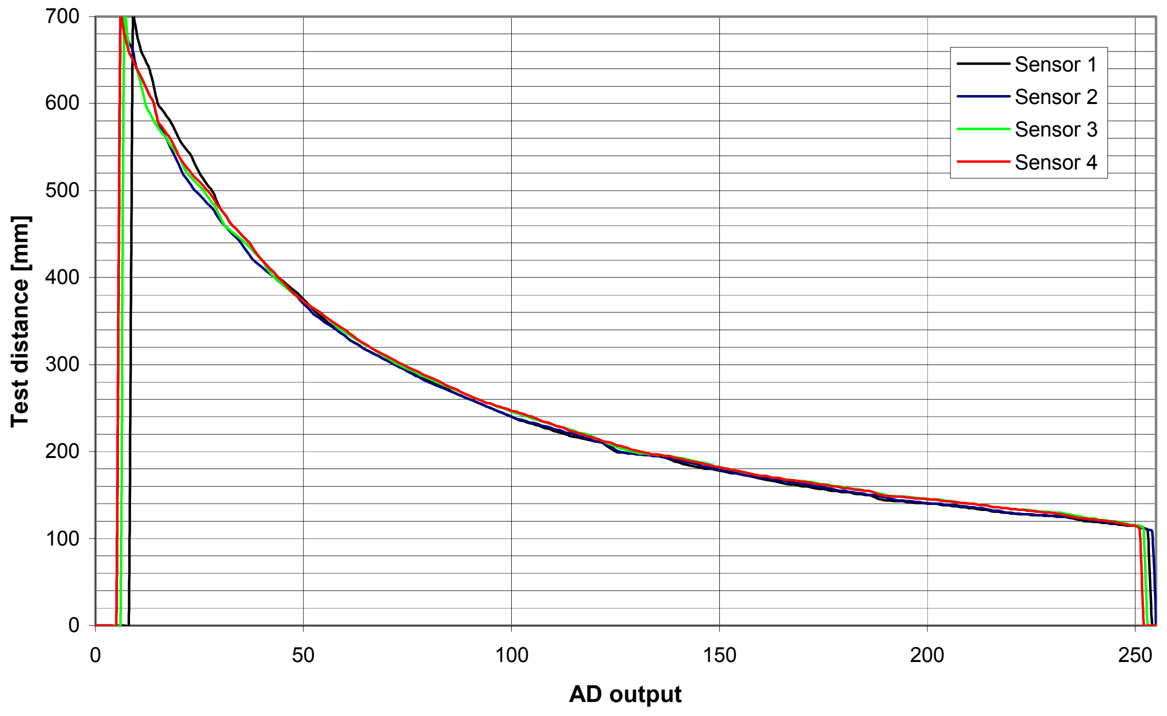


Fig. 4.6 – Characteristics of four different Sharp GP2D12 sensors

As one could deduce from Fig. 4.6, the difference between the sensors is negligible. This observation leads us to implement a unique lookup table that is the average of the four sensors:

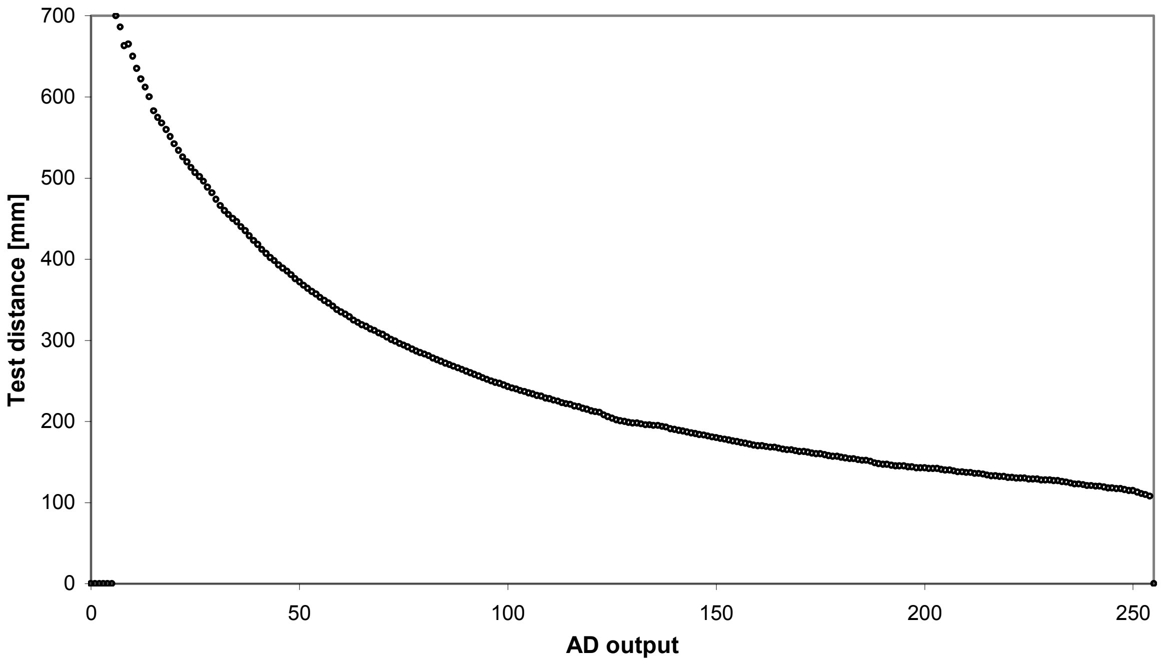


Fig. 4.7 – Average characteristic of the Sharp GP2D12 sensors

Of course, the 40 reference distances for each sensor do not directly provide the required 255 values for the lookup table[24]. The BLHSupervisor automatically computes a linear approximation between each pair of successive points. On the following graph, only the actually measured points (for one of the four sensors) and their corresponding standard deviation are shown.

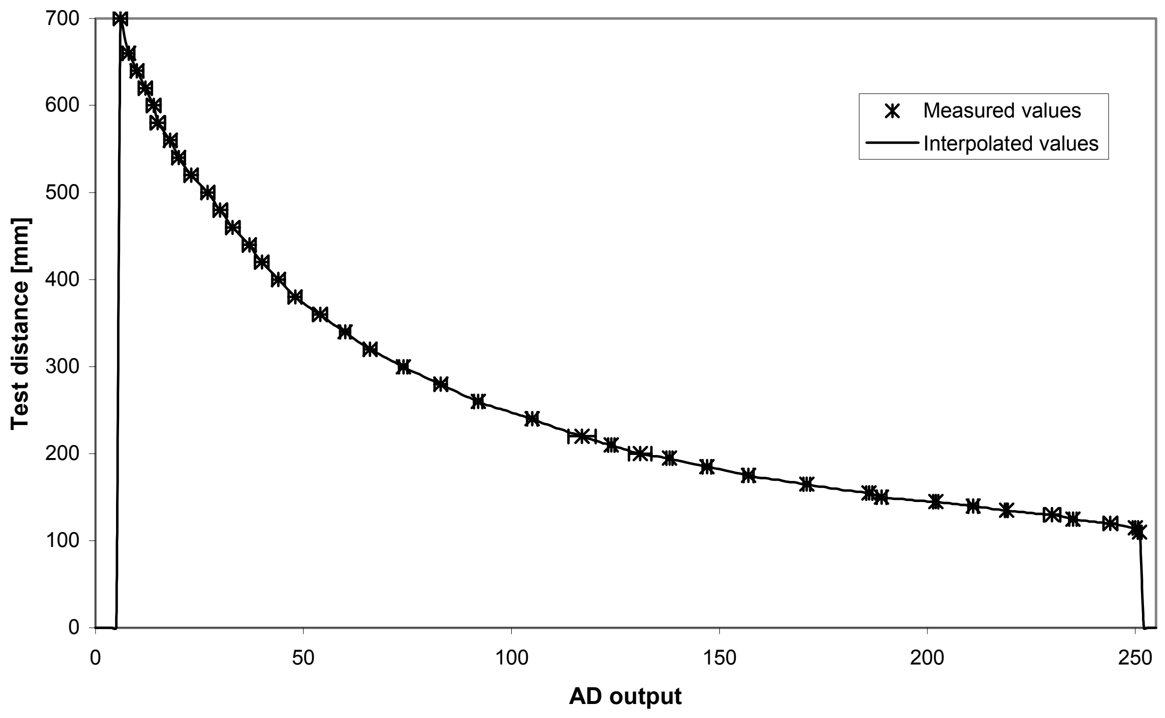


Fig. 4.8 – Set of 40 measured values and their corresponding standard deviation

Based on this graph, a rapid evaluation of the precision can be made. It is easy to understand that the values above 700mm are almost unusable. At 600mm, the precision is only 20mm and becomes better with closer distance (about 2mm at 200mm).

4.3 Position of the sensors on the body

This section aims to discuss the main parameters that have to be chosen for the installation of the sensors on the robot. For this purpose, some dimensions of the existing machine will be used. For sake of clarity a lot of simplifications will be made that help to bring out the most significant parameters. The disposition of the four sensors is not determined a priori and will be deduced from the discussion and the end of the present section. For sake of simplicity (also for the sensing routine), we assume that they are evenly distributed around the hip, thus forming rectangle centered at the hip-joint.

4.3.1 Geometric considerations

The sensors are placed under the body, vertically oriented toward the floor such that a total symmetry exists when the body swings around the hip (reference point). As the four sensors form a rectangle, a cross section through two opposite sensors is enough in order to analyze the geometry of the system. Therefore the discussion will be made in the two-dimensional case.

[24] Name in the program: `AD2Dist`

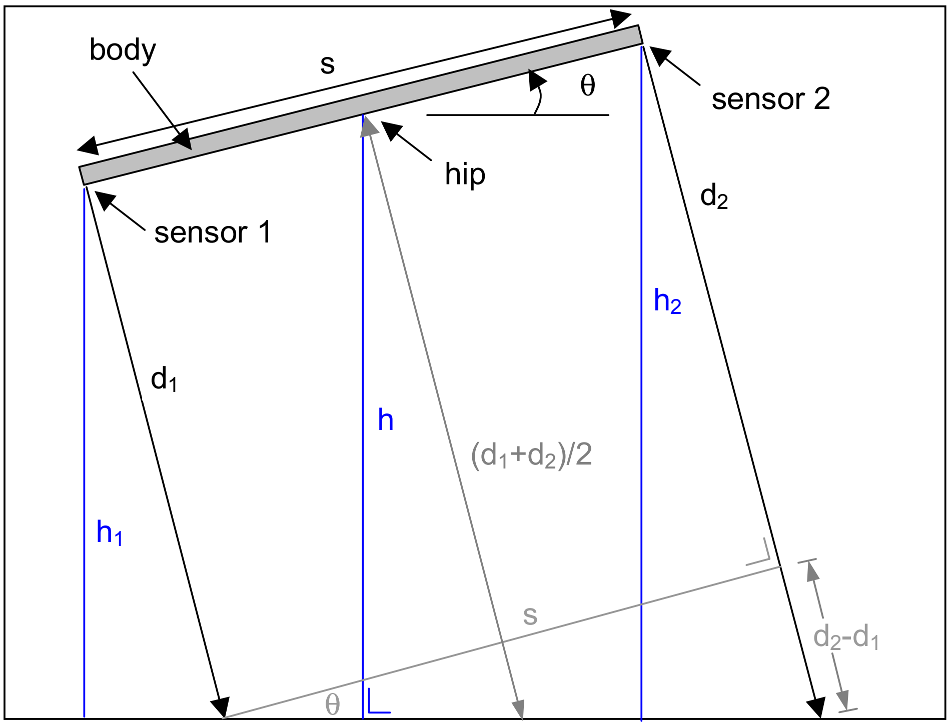


Fig. 4.9 - Variables used in the discussion of sensors arrangement

The span between two opposite sensors is denoted by s. The suffixes after the symbols represent the index of the sensor. For a systematic description of the symbols, please consult the Appendix 8.1.

Based on simple geometric considerations, the following equations are worked out.

- **Body angle θ**

The body angle is given by:

$$\theta(d_1, d_2) = \arctan(\frac{d_2 - d_1}{s}) \qquad \textbf{\textit{Eq. 4.1}}$$

- **Body height h**

Once the body angle θ is known, the height of the body may be calculated as follows:

$$h = \frac{d_1 + d_2}{2} \cdot \cos(\theta) \qquad \textbf{\textit{Eq. 4.2}}$$

- **Range of measured distances**

Another interesting thing to know is the distances seen by the sensors for a given height and body attitude:

$$d_1(h,\theta,s) = \frac{h}{\cos\theta} - \frac{s}{2}\tan\theta$$ ***Eq. 4.3***

$$d_2(h,\theta,s) = \frac{h}{\cos\theta} + \frac{s}{2}\tan\theta$$ ***Eq. 4.4***

Here is a graph of these two functions, with h=250mm and s=230mm:

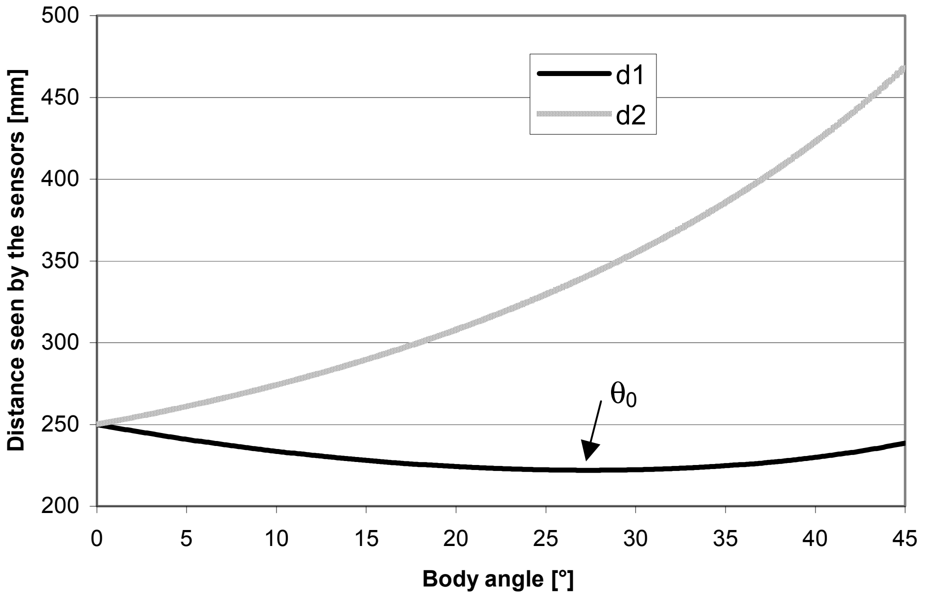


Fig. 4.10 – Range of measured distances for h=250mm and s=230mm

If we want to calculate the $d_{i,max}$ value for a given s, h_{max} and θ_{max} it is straightforward because this function is monotonic:

$$d_{i,max} = \frac{h_{max}}{\cos\theta_{max}} + \frac{s}{2}\tan\theta_{max}$$ ***Eq. 4.5***

The calculation of $d_{i,min}$ is a bit more complicated. We must take the first derivative of Eq. 4.3 with respect to θ:

$$d_i^{'}(\theta) = \frac{2h_{min}\sin\theta - s}{\cos^2\theta}$$ ***Eq. 4.6***

As we are looking for the minimum of equation Eq. 4.6, we must find the value of θ for which equation (8) is equal to zero, assuming cos(θ) different from zero because θ<90° in our case. Let's call this special angle θ_0:

$$\theta_0(h_{min},s) = \arcsin\frac{s}{2h_{min}} \quad \text{if } \theta_0<\theta_{max}$$ ***Eq. 4.7***

We must be careful with this last formula, because θ_0 may be greater that θ_{max} in this case, θ_0 must be replaced by $\theta_{max.}$

Finally, we have:

$$d_{i,min} = \frac{h_{min}}{\cos\theta_0} - \frac{s}{2}\tan\theta_0$$ **Eq. 4.8**

4.3.2 Discussion

Based on these equations, the influence of three significant parameters on the distance seen by the sensors will be discussed. For sake of simplicity, h_{min} will be approximated[25] by the leg length l. Its value is about 200mm.

➢ **Significant parameters**

The following parameters determine the range of the measured distances:

- $\mathbf{h_{max}}$ depends essentially on the energy stored in the leg during flight. It varies also with the leg angle ϕ, but for this discussion, we assume that the robot is hopping in place, thus ϕ = 0°. Notice that the time of flight T_f also depends on h_{max} (see § 5.1.1). And, θ_{max} depends on the time of flight T_f. The smaller the time of flight is, the minimum the body swing will be, because of the constant angle velocities during flight.
- $\boldsymbol{\theta_{max}}$ is not really predictable before experimentation in real gravity without damping torques applied on the body. I think we must at least allow for a value of 30°, if not more.
- Finally, **s**, the span between the sensors is a quite open parameter. However, there are some mechanical limitations: the width of the body and the clearance for the leg (the leg must not enter the view of the sensors). The size of the body is 432 x 254mm (17 x 10inches). Concerning the leg clearance, we can make a first approximation with a maximum β angle of 30°. This defines a circle with a radius of l.sin(β)≈250.sin(30)=125mm.

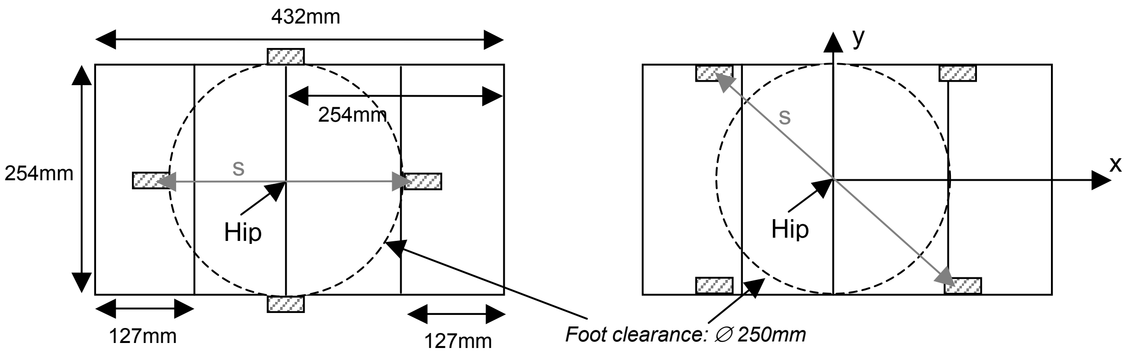


Fig. 4.11 - Top view of the 3D Bow Leg Hopper for sensor arrangement

So the value of the s parameter must be at least 250mm (configuration shown on the left). The maximum value of s, if we remain within the surface of the current body, is about 360mm (configuration shown on the right).

➢ **Range of the distances seen by the sensors**

Now that these parameters are defined, we can finally discuss their implications on the distances seen by the sensors. Finally this will help us to select the parameter values that will keep the sensors working in their useable range.

[25] More precisely, h_{min} depends on the leg compression and the maximum leg angle with respect to *R*: ϕ_{max}.

The following 3D graph shows the **minimum distances seen by the sensors** as a function of the span between the sensors and the minimum height of the body: $d_{i,min}(s,h_{min})$ (from Eq. 4.8). Note that θ_0 is an implicit parameter and that the minimum measurable distance is about 108mm (represented on the graph).

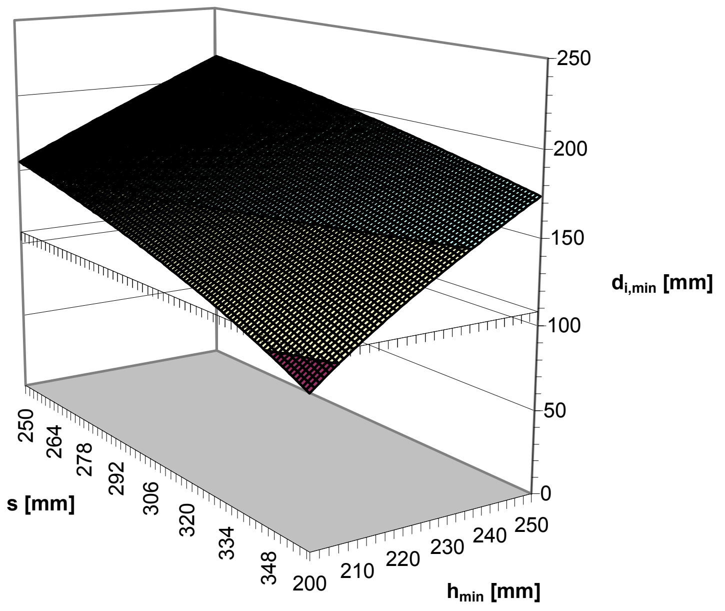


Fig. 4.12 – Minimum distances seen by the sensors

The previous graph doesn't take care of the notice after Eq. 4.7. If we limit θ_0 at θ_{max}=30°, the situation gets better because θ_0 is almost always greater than 30°:

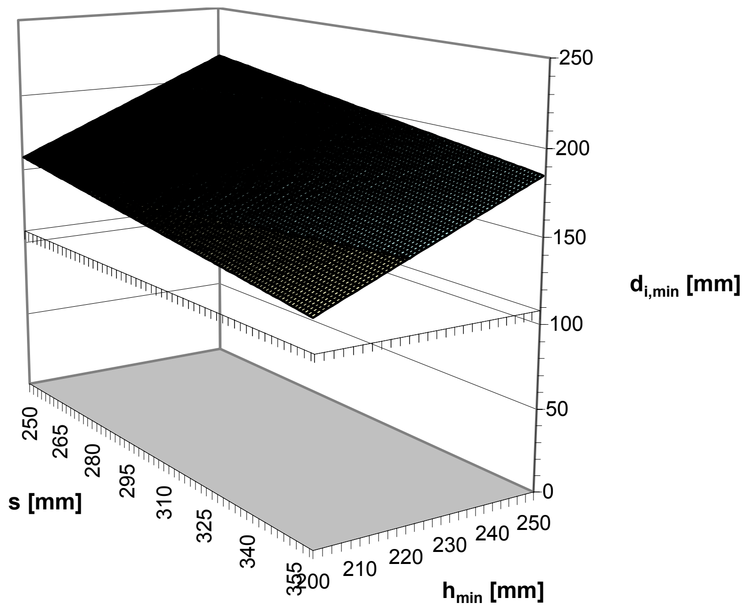


Fig. 4.13 - Minimum distances seen by the sensors with θ_0<30°

So, there is no great problem about the minimum distances and we can be sure that the lower limit of the sensor would never be reached under normal conditions.

What's about the **maximum distances**? For $d_{i,max}$ at the apex of the flight trajectory, we can get an idea of the influence of the body attitude (θ_{max}) and the apex height (h_{max}). Let's assume s=360mm, the worst case for the sensors, in order to plot the surface of Eq. 4.5.

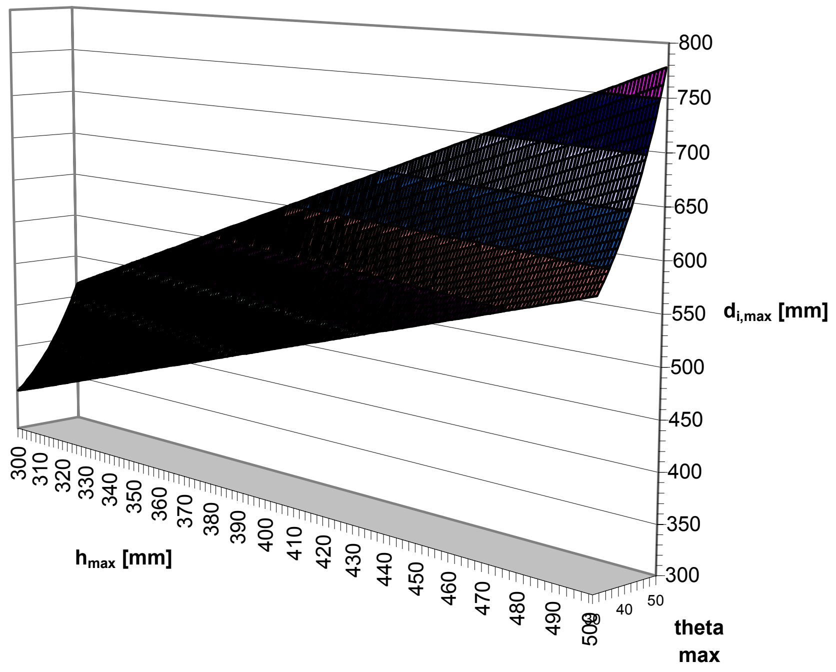


Fig. 4.14 – Maximum distances seen by the sensors

This time, we must keep in mind that the distances above 600 or 650mm are not accurately rendered by the Sharp sensors. Moreover, at 600mm the precision is ±10mm and become better with lower distances (see § 4.2.3).

4.3.3 Conclusion

If s, the span between two opposite sensors, is large, the computed angles will be more precise but the range of the measured distances becomes also larger. Moreover, if s is too small, the leg clearance may be not enough.

Another thing to keep in mind is that we are not forced to have useable distances all the time. For example, distances during stance are not very useful. And if the hopper jumps too high, we could be content with a set of values just after liftoff and/or just before touchdown. As we will see later on, the two gyroscopes may be used to make a quite good interpolation. Thus having a few but as accurate as possible body attitudes is the best.

For this reason, and also because it improves the leg clearance, the arrangement shown on the right in Fig. 4.11 has been chosen. The sensors are on the corners of a square. The sides of the square are 235mm long and the diagonals (same as s) 332mm.

This configuration introduces a bit more complicated calculation because the sensors are not directly on the roll and pitch axis. But this is a detail since all we have to do is to simulate four sensors exactly positioned on the axis by averaging the adjacent pairs of real sensors.

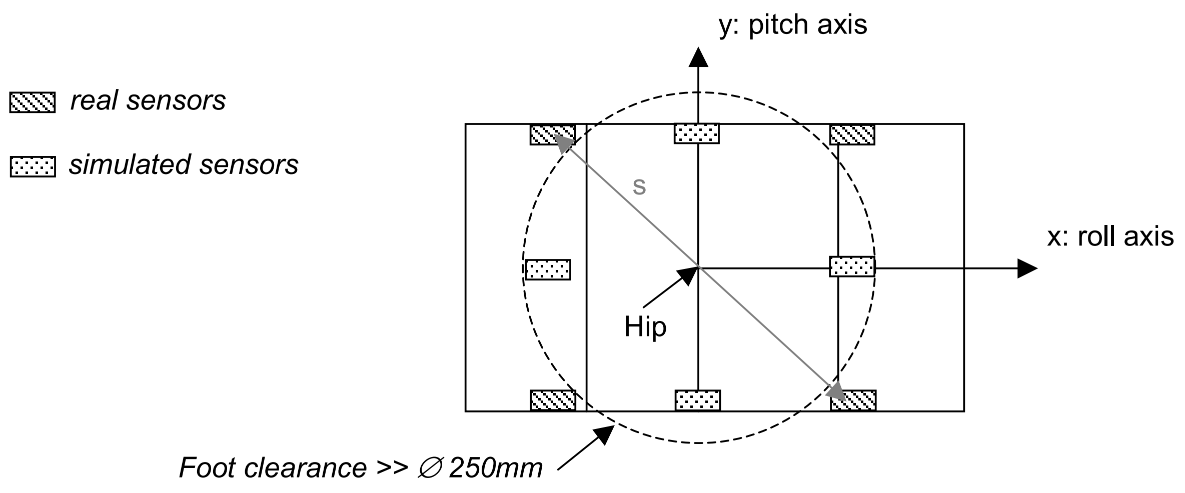


Fig. 4.15 – Final arrangement of the sensors under the body

4.4 Dynamic distance measurement

4.4.1 Sensor output over time

The analog output of the Sharp GP2D12 sensors moves in voltage steps when the measured distance changes. These steps last about 40ms. See [wwwSharp] for complete data sheet.

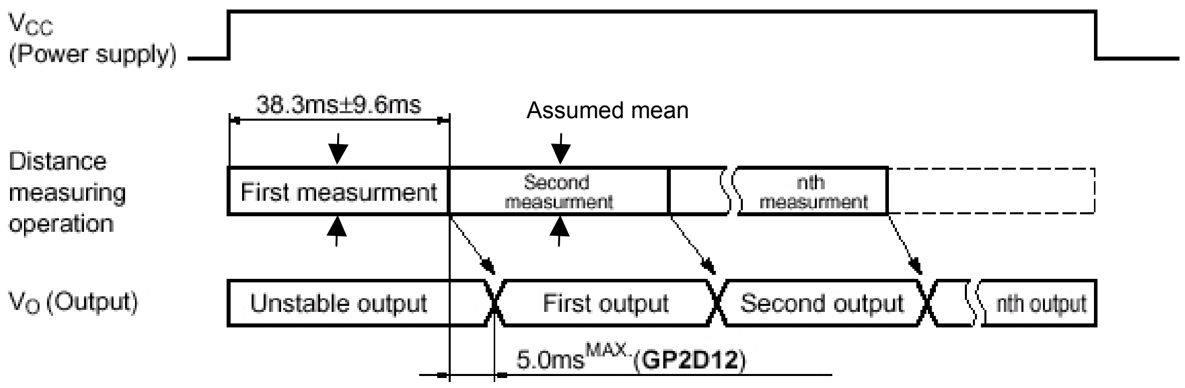


Fig. 4.16 extracted from [wwwSharp] – Sharp GP2D12 operation over time

This behavior is certainly due to the fact that these sensors process an output signal by internally averaging successive values. Let’s assume that the best way to interpret this signal is to take the step level as the value measured at its middle. Because there is some noise on the sensor output, we can get an improved value of the distance by averaging the output along the step.

Fig. 4.17 shows the typical sensor output during a fast movement of the target object.

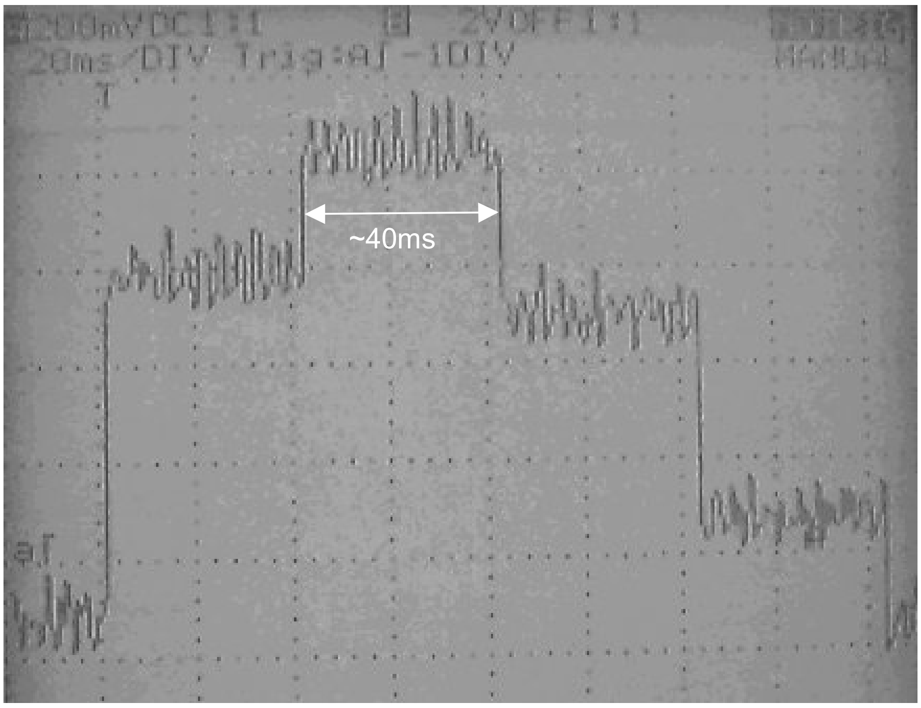


Fig. 4.17 – Sharp sensor output during dynamic measurement (200mV and 20ms per division)

The noise on the steps can reach about ±100 mV.

What is more interesting for the following is the same signal seen by the microcontroller. Fig. 4.18 illustrates this. Note that the movement is not parabolic: these measures are not taken during real jumps of the robot.

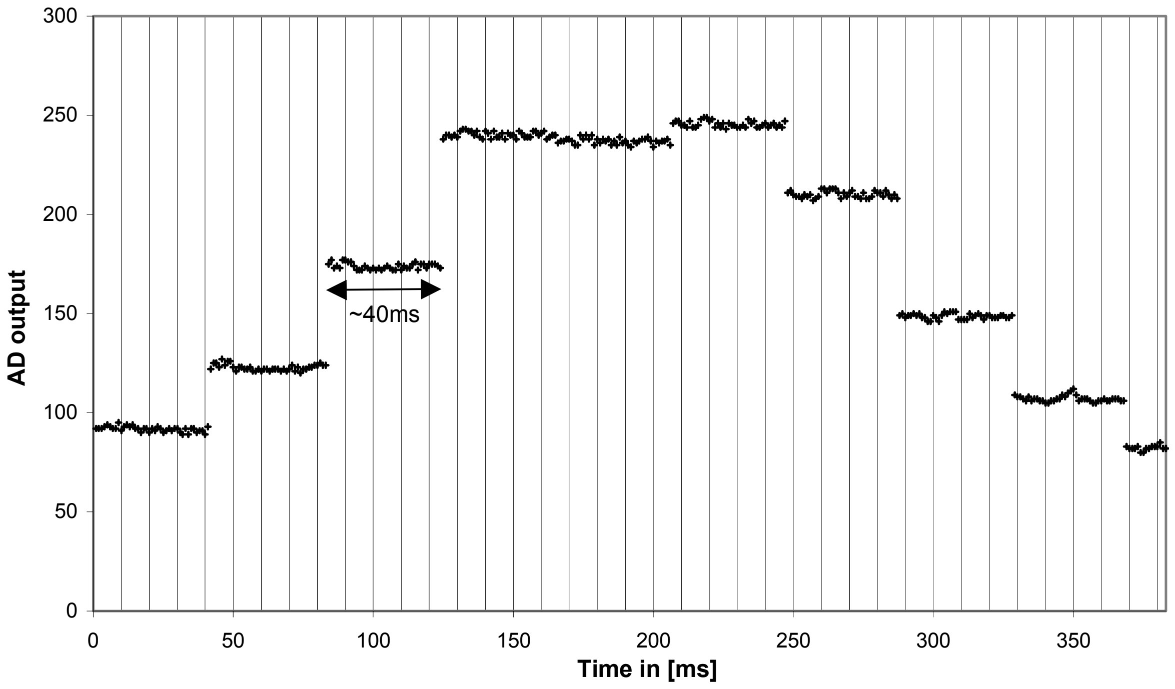


Fig. 4.18 - Sharp sensor output seen by the microcontroller during dynamic measurement

The reading frequency of the converter is 1kHz. Therefore the processor can have an updated value every 1ms and thus about 40 values per step. In the microcontroller, this number is defined as `MEAN_STEP_DURATION`. These 40 values will be averaged over time in order to obtain a good approximation of the step level without noise (this procedure acts like a digital low-pass filter). To achieve this, the points belonging to the same steps must be recognized and clustered.

4.4.2 Clustering the values belonging to the same step

There are essentially two criteria to achieve this clustering. The first is certainly the difference between successive data. If the difference is greater than a certain threshold, the processor can know that it is the next step. The second one is the maximum time allowable for a step. If two of them are merged, we can end the set of successive values belonging to the same step by knowing the maximum length of this set. The maximum number of conversion during a step is 43. So if there is no significant transition after 43 readings, we can assume that future data are part of the next step. If some data are not well classified, in this case, this is not so important because it will not significantly change the mean value of the steps.

If all the steps are at the same height (static case), this clustering process will lose the synchronization. But the 3D Bow Leg Hopper is certainly not built to stay in place. Moreover, as soon as a transition appears, the algorithm will automatically synchronize again.

This algorithm has been implemented in the microcontroller. The computed means of the steps are represented in black on Fig. 4.19. Please note that the clustering is applied to the rough sensor values and not to the distances after conversion through the lookup table. This is algorithm is intended to filter the noise of the sensor output, which is constant over the full scale of distances. If the successive values from the sensor were first converted into distances, the noise would have not always the same magnitude.

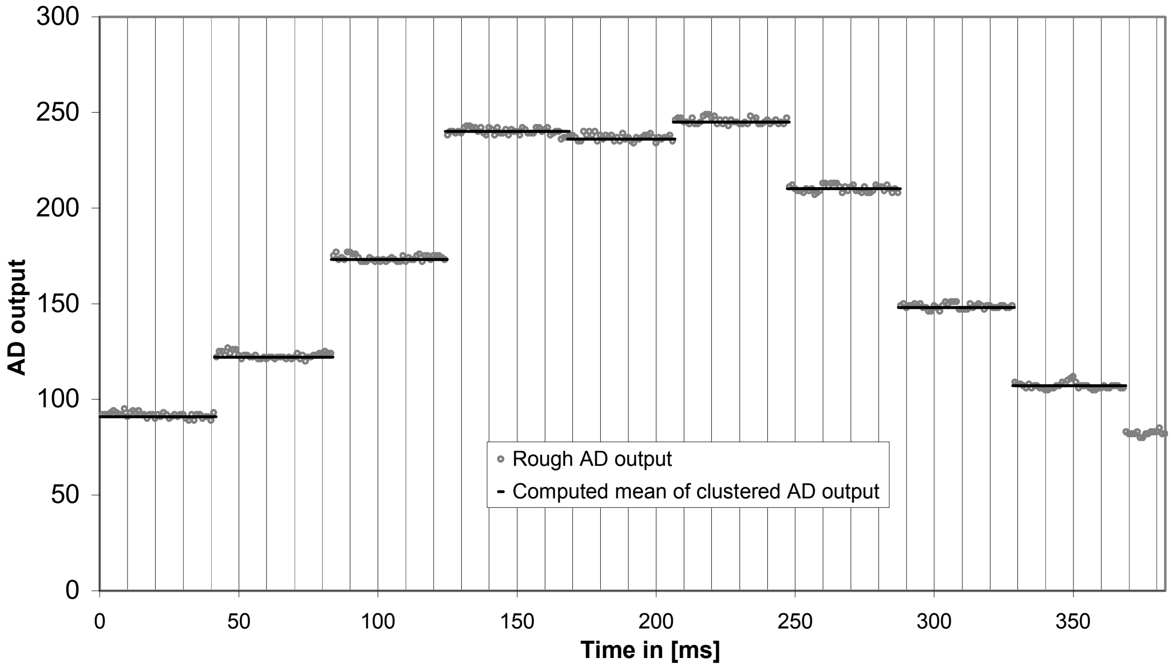


Fig. 4.19 – Result of the clustering process

The threshold for the difference is set to 15 and the maximum number of points belonging to the same step was 43. In the following, the former value will be denoted by `THRESHOLD` and the latter by `MAX_STEP_DURATION`.

We can see that there was not enough difference between the fourth and the fifth steps. Therefore the 43 first points have been allocated to the fourth step. And the last points (actually 38) before the next recognized transition are part of the fifth step.

Note that the computation of this average does not require 40 or 43 registers in memory. A sum variable on 16-bit is updated at each reading of the converter. When the end of a cluster is detected, the sum is divided[26] by the number of data in the set.

Each computed average will be then transformed into a distance via the lookup table (see § 4.2.3). This distance is time stamped with the middle of the previous step.

4.4.3 Summary

To sum up, we have now an algorithm[27] (described in the previous paragraph), which is able to process the rough sensor data at 1kHz and output an enhanced distance value about[28] each 40ms i.e. at a frequency of 25Hz, which is the update rate of the Sharp GP2D12 sensors.

On the following graph (Fig. 4.20), the whole process is revealed through real values coming from the microcontroller in real time thanks to the BLHSupervisor.

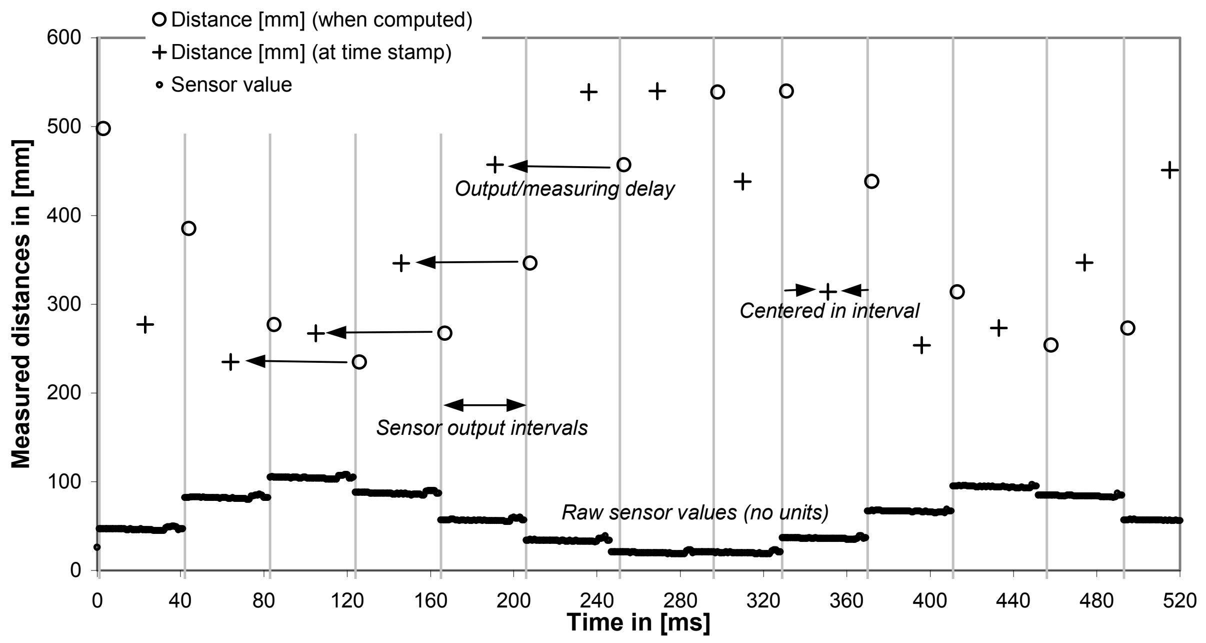


Fig. 4.20 – Clustering process and resulting distances

A point represents each acquired sensor value, every millisecond. The vertical gray lines indicate the moment when the end of a step is detected: whether a jump higher than a specified threshold or the overrun of the maximum successive values allowable for one step. The little circles symbolize the distances at the moment they are calculated. The crosses are the same values but plotted at the position they are time stamped thus 60ms before their calculation[29].

[26] This is a quite expensive operation because the division does not belong to the basic set of operations of the microcontroller. The C compiler has some integrated routines to compensate for that sort of gaps. The time consumed for such a division (int16/int8) is approximately 17µs.

[27] Actually, this process acts like a digital low pass filter in the scope of each step, after having clustered them. If an analog low pass filter is applied in order to diminish the noise, it seriously diminishes the dynamic capability of the sensor.

[28] This period may fluctuate a little bit, because the clustering process cannot be always reliable, especially when no transitions are present in the signal (see Fig. 4.20).

[29] All this work is accomplished within a function named `ClusterAndSlope()` which is call one time for each sensors, each millisecond.

4.5 Computing body attitude

Now the concern is twofold:

1. The distances of the four sensors will be computed asynchronously because there is no way to externally trigger these Sharp sensors.
2. As discussed before, the distances will be worked out with a delay of 3/2 the length of the step, i.e. about 60ms[30].

The following graph illustrates the problem of non-synchronization between the sensors.

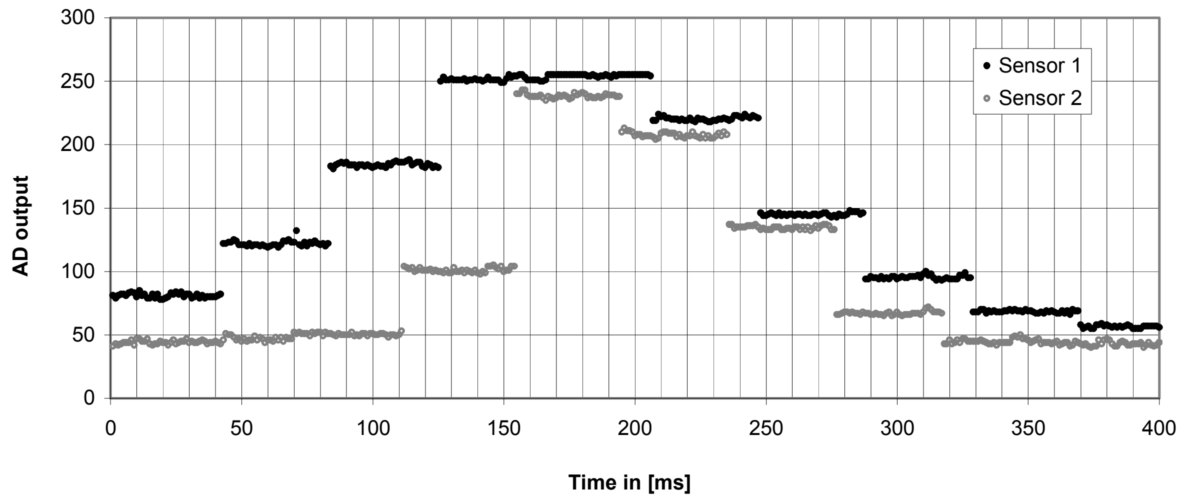


Fig. 4.21 – The update rates of the sensors are not synchronous

After that, we want to estimate the body attitude based on the four sensors, thus four asynchronous and delayed distances. In the following sections, I will show a solution to this problem. Please, keep in mind that the algorithm must be implemented on a simple 8-bit microcontroller, which has not plethora of RAM memory.

Since the distances are available asynchronously, an asynchronous routine should be able to estimate a simultaneous distance for each of the four sensors. This information allows computing a good body attitude and height.

Because of the lack of memory, it is not possible to store the recent history of the distances for the four sensors. To alleviate this problem, at the moment a new distance value is computed, the slope between the new value and the previous one is stored. For good understanding, we have an array in RAM memory with a place for the four distances and for their corresponding slopes.

Then, an asynchronous process[31] that runs at exactly 25Hz[32] is able to deduce simultaneous distances from the above-mentioned array based on the four distances and their slopes. In the following graph, which is a completed version of Fig. 4.20, the vertical black lines embody this asynchronous process, each 40ms. Note that the slopes between two successive distances are plotted at their time stamp and their value in [mm/ms] is multiplied by 32 for precision reasons[33].

[30] This number is represented by `TS_DELAY = MEAN_STEP_DURATION*3/2`
[31] This work is carried out by a function named `DTildeComputation()`.
[32] Loop that starts every 40ms, based on the millisecond timer.
[33] Since the microcontroller handles only integer variables, it is necessary always trying to use the full span of the 255 values allowable on 8-bit.

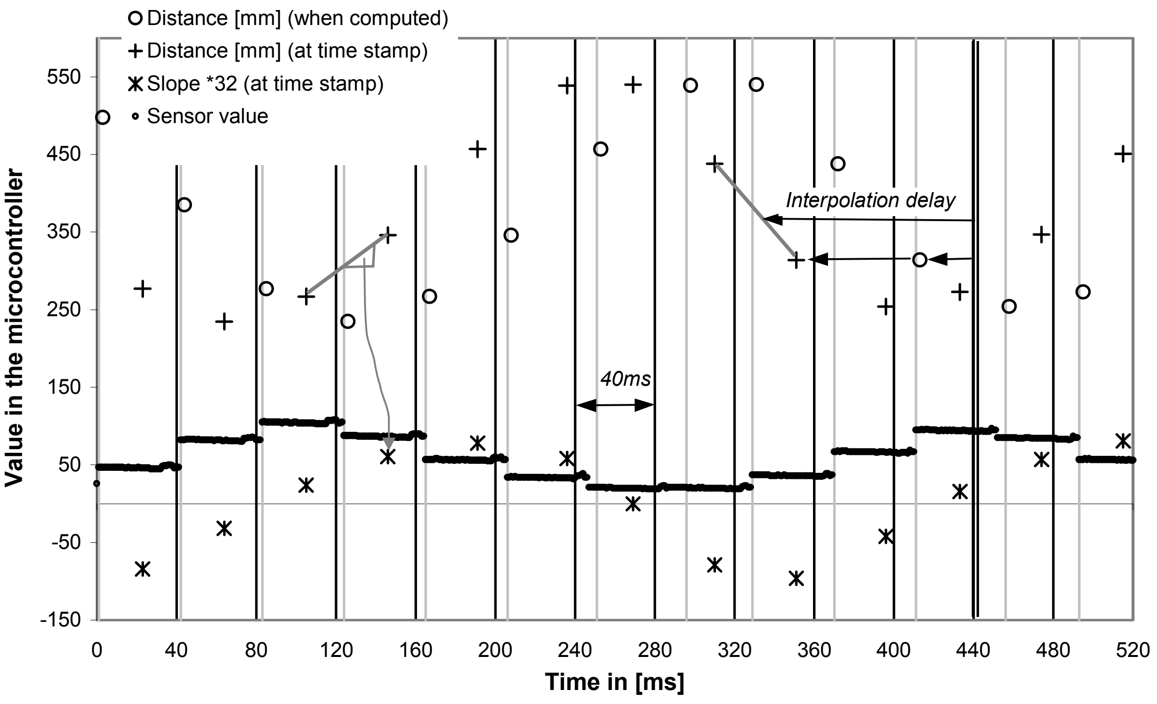


Fig. 4.22 – The vertical black lines embody the asynchronous process @ 25Hz

Keep in mind that the distances and their corresponding slopes exist only until the following ones replace them. Moreover, the time stamped values are not available before they are computed (!). For these reasons, at each 40ms (asynchronous process) a moment will be chosen which guaranties to be situated before all the time stamped distances and thus in the region where the stored slopes are a good tool in order to approximate (by linear approximation) the real distances under the sensors. This moment is 103ms[34] before the computation time.

In doing so, it is possible to obtain four simultaneous estimated distances. Based on them, the roll (θ_r) and pitch (θ_p) angles can be computed, using Eq. 4.1. Finally, the height of the body center of mass is deduced from Eq. 4.2. The cost of this process is the delay of about 100ms between these estimations and their availability for control. And it is essentially due to the lack of the possibility to externally trigger the Sharp GP2D12 sensors.

[34] `TS_DELAY + MAX_STEP_DURATION = 60 + 43 = 103ms`

5 Control

5.1 Modeling

In the following, some basic relations are presented, assuming the Bow Leg Hopper bouncing in place in equilibrium.

5.1.1 Flight

As during flight the COM of the body follows a ballistic trajectory, some simple equations can easily be deduced. Let's take Δh as the vertical distance the hopper falls from apex to impact and assume the leg is vertical. We have:

$$\Delta h = h_a - l = \frac{1}{2} g \left(\frac{T_f}{2} \right)^2 \qquad \textbf{\textit{Eq. 5.1}}$$

Where T_f is the time of flight, h_a the height at apex and g the effective gravitational acceleration.

The vertical velocity at touchdown is:

$$\dot{h}_t = -\sqrt{2g\Delta h} \qquad \textbf{\textit{Eq. 5.2}}$$

The flight time is given by:

$$T_f = \sqrt{\frac{8\Delta h}{g}} \qquad \textbf{\textit{Eq. 5.3}}$$

See Table 6.1 and Table 6.2 for some values of time of flight for different hopping amplitude Δh.

5.1.2 Stance

The following section is directly adapted from [Zeg99], Appendix B.1. Please, refer to the thesis for more details.

The leg of the robot is idealized as a massless constant force spring (F_0) with non-ideal restitution. This leg does dissipate energy but the thrust mechanism is assumed to exactly compensate the losses.

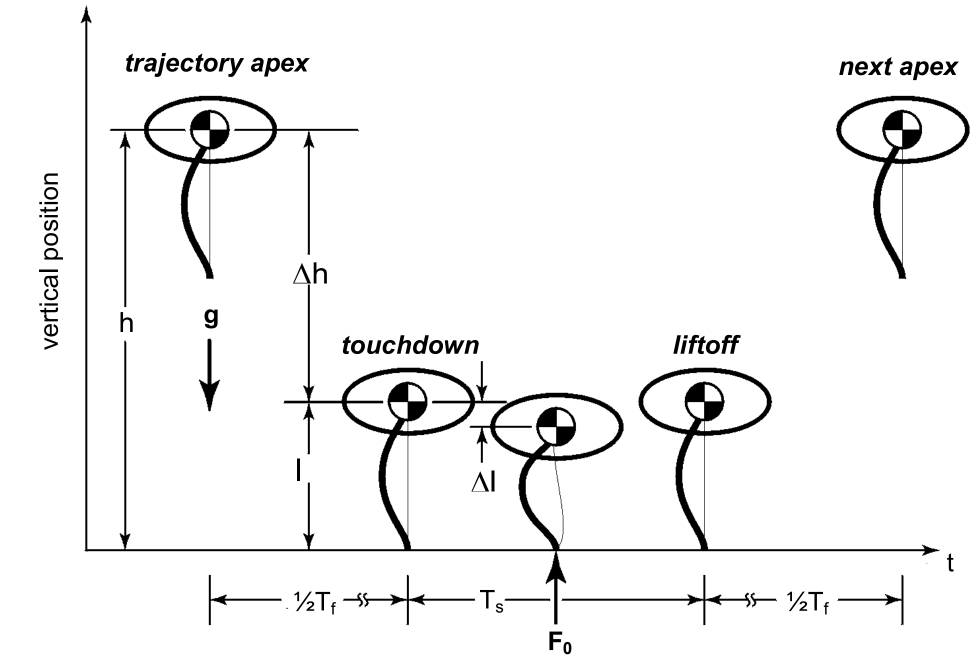


Fig. 5.1 (adapted from [Zeg99]) – Variables used in stance dynamics discussion

Assuming that the momentum at impact is exactly reversed by the constant force leg spring applied during stance, the stance time can be expressed like this (equation B.5 in [Zeg99]):

$$T_s = \frac{-2M\dot{h}_t}{F_0} = \frac{M}{F_0}\sqrt{8g\Delta h} \qquad \textit{Eq. 5.4}$$

Where M is the mass of the robot (i.e. the body, in this case) and F_0 the assumed constant spring force of the leg.

In order to deduce the change in leg length as a function of equilibrium hopping, some considerations about energy transfers have to be made. The leg compresses to store kinetic energy in leg potential energy. The energy stored was gained from the potential change from the highest trajectory point to the point of maximum compression. Finally, we obtain an estimation of Δl as a function of Δh (equation B.13 in [Zeg99]):

$$\Delta l = \frac{-Mg\Delta h}{F_0 + Mg} \qquad \textit{Eq. 5.5}$$

5.1.3 Body attitude

As said before (see § 2.1.1), the body orientation of the Bow Leg Hopper should be passively stabilized. The planar hopper has successfully demonstrated this effect[35]. Following is a quick analysis of the phenomena. Fig. 5.2 shows the forces applied to the body during stance, assuming a constant force provided by the bow leg (see § 2.3.1) and a vertical position of the leg[36] thus $\beta=-\theta$.

[35] However, we do not exactly know the importance of the damping moment due to the bearing between the boom and the body.

[36] The first simple controller does approximately this job: keeping the leg vertical assuming the robot stays in place. However, with constant velocity, the leg will be vertical on average because sweeping fore and aft during stance (see [Rai86], Figure 2.10 or [Zeg99], § 3.3.2).

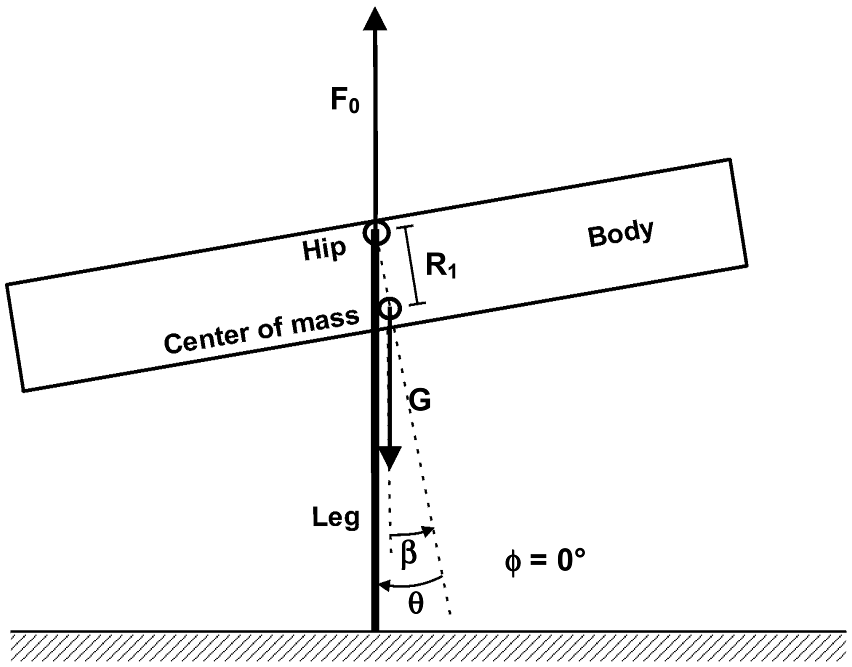


Fig. 5.2 – Forces on the body during stance

The distance between the hip and the COM is denoted R_1. $\mathbf{F_0}$ is the force applied to the hip by the leg during stance. **G** is the force due to the gravity.

The torque at the COM during stance can thus be approximated by:

$$\tau_s(t) = F_0 \cdot R_1 \sin(\theta(t)) \qquad \textit{Eq. 5.6}$$

This torque tends to bring the body back to a horizontal position, like a pendulum. The angular body acceleration during stance is given by:

$$\ddot{\theta}(t) = \frac{\tau_s(t)}{J} \qquad \textit{during stance} \qquad \textit{Eq. 5.7}$$

where J is the moment of inertia of the body[37].

Of course, this stabilizing torque exists only during stance. As soon as the leg leaves the ground, the spring force disappears and the body will maintain its angular velocity around the COM until next bounce:

$$\dot{\theta}_r, \dot{\theta}_p = \text{const} \qquad \textit{during flight} \qquad \textit{Eq. 5.8}$$

What is quite important in our case is to take care of not exciting this body swing oscillation by the jumping frequency. During stance, the resonant frequency of such a pendulum is simply given by:

$$\omega_s = \sqrt{\frac{F_0 R_1}{J}} \qquad \textit{Eq. 5.9}$$

[37] Actually, J is the moment of inertia around one of the principal axes. In our case, we can assume the principal axes are co-incident with the axis of the frame *B*: roll, pitch and yaw axis. So in this 2D analysis, J represents J_r or J_p.

To figure out the average body swing frequency during normal operations, we can assume that the average value of F_o is equal to Mg, on many hopping cycles[38]. This assumption leads to a sort of average of the body swing resonant frequency over time:

$$\bar{\omega} = \sqrt{\frac{MgR_1}{J}} \qquad \textbf{\textit{Eq. 5.10}}$$

This value may also be found from experiments as we will see in § 6.2. The resonant frequency of the body should always be much less than the jumping frequency to avoid excitation by the hopping cycles.

5.2 Control of running

5.2.1 Bow Leg Hopper control

The Bow Leg mechanical design permits only one control cycle per bounce unlike Raibert machines (see § 2.1.2). This determines the properties of controller, which takes the general form of the following function[39]:

$$\begin{bmatrix} \vec{\phi}_{t,n+1} \\ \xi_{t,n+1} \end{bmatrix} = f(\vec{x}_{f,n}, h_{f,n}, \dot{\vec{x}}_{f,n}, \vec{\theta}_{f,n}, \dot{\vec{\theta}}_{f,n}) \qquad \textbf{\textit{Eq. 5.11}}$$

- Input: the information about flight trajectory preceding the impact (apex position and lateral velocity), the body attitude and angular velocity. Some of these parameters are constant during flight, other must be estimated at a precise moment, for instance at apex.
- Output: simply the leg angle and the amount of energy that must be stored in the leg.

An in-depth discussion about the Bow Leg Hopper control takes place in the fourth chapter of [Zeg99] thus I shall not expose it in more details here.

5.2.2 Reduced control for initial assessment

As said before, different forms of a simpler controller for initial tests has been developed, in which no care is taken of the lateral velocity. This controller function takes the following form:

$$\beta_{t,n+1} = f(h_{f,n}, \vec{\theta}_{f,n}, \dot{\vec{\theta}}_{f,n}) \quad \text{with } \phi_{n+1}=0 \text{ and } \xi=\text{const} \qquad \textbf{\textit{Eq. 5.12}}$$

The leg angle in the world coordinates at touchdown (n+1), $\phi_{t,n+1}$, is maintained equal to zero. To reach this, after Eq. 2.1, the controller must position the leg angle with respect to the body, $\beta_{t,n+1}$, equal to $-\theta_{t,n+1}$, which is the body attitude at impact (n+1). To accomplish this task, the controller will take into account one or several of the following parameters: height of COM at apex, body attitude, body angular velocity and time from liftoff to apex.

Notice that this control is marginally stable for velocity. Small lateral velocity errors will accumulate, and be compensated for by human driver.

➢ **Keep the leg always vertical**

The more straightforward way of implementing such a controller is to keep the leg always vertical during flight. Thus, only the body attitude, θ, has to be known.

[38] F_0=0 during flight and F_0 >> Mg during stance.
[39] The vector notation is employed here to make a straightforward generalization from 2D to the 3D case. **x** represents the lateral position. For information about symbols, please refer to Appendix 8.1.

But, as demonstrated at the end of Ch. 4, the information of body attitude is available with a latency of about 100ms, which is quite important relative to the jumping period: 200 to 700ms depending on the gravity[40]. In order to compensate for this delay, the two gyros can be used. Their latency is much less: about 5ms, which is the rate of the input PWM.

This controller has been implemented on the real robot and the results of some static tests may be found in § 6.3. The main concern is that the body attitude values used for positioning the leg at impact are based on measurement made 100ms ago, which corresponds to a quite high position of the robot (near the apex) and thus a less accurate estimation of pitch and roll angles.

- **Estimate the body attitude based on gyroscope interpolation**

An alternate way of implementation in order to reduce this problem is to base the body attitude estimation on some good values taken just after takeoff, when the distances given by the sensors are more accurate. Then the roll and pitch angles can be interpolated based on the gyros until impact. This possibility does not need any information about the height of the robot during flight, too.

In general, the signal provided by the gyros is less noisy than the angles estimated with the range finders (see § 6.2). So this second solution[41] may be better than the previous one but requires a good calibration of the gyros, which is not so easy to achieve.

- **Estimate the body attitude at next touch down by parabola fitting**

The idea here is to use the knowledge of the ballistic trajectory during flight to deduce the next moment of touchdown, t_t. Assuming the processor knows the liftoff time by reading the foot sensor, it may compute the the apex time t_a based on a parabola fitting and then predict the touchdown time t_t:

$$t_a - t_l = T_r = T_{fall} = t_t - t_a \qquad \textit{\textbf{Eq. 5.13}}$$

Here is the equation of the body height along the time:

$$h(t) = at^2 + bt + c \qquad \textit{\textbf{Eq. 5.14}}$$

Actually, we don't need a total parabola fitting but only the time of its apex. That means the time at which the first derivative of the previous equation is equal to zero. For this calculation we don't need to know the vertical position of the parabola (i.e. the term c). Moreover, the parameter a is known: a=1/2g. So we have:

$$\dot{h}(t) = gt + b \qquad \textit{\textbf{Eq. 5.15}}$$

$$\dot{h}(t_a) = gt_a + b = 0 \quad \Rightarrow t_a = -\frac{b}{g} \qquad \textit{\textbf{Eq. 5.16}}$$

Thus, the goal is to compute the value of the parameters b. To achieve this, the slope value of the parabola at a precise moment is enough. This value can be approximated in taking the difference between two successive heights:

$$\tilde{\dot{h}}(t_{tilde}) = \frac{h(t_{\tilde{k}+1}) - h(t_{\tilde{k}})}{t_{\tilde{k}+1} - t_{\tilde{k}}} \text{ where } t_{tilde} = \frac{t_{\tilde{k}} + t_{\tilde{k}+1}}{2} \qquad \textit{\textbf{Eq. 5.17}}$$

Finally, we have:

[40] See § 6.1 for more information about experiments under low gravity.
[41] Its implementation is currently in progress.

$$b \cong \tilde{\dot{h}}(t_{\text{tilde}}) - g \cdot t_{\text{tilde}} \qquad \textbf{\textit{Eq. 5.18}}$$

And then the value of t_a is easily calculated from Eq. 5.16.

Once the moment of next touchdown is known, the body attitude at touch down may be calculated based on the current body angle velocity. Notice that this is possible only if the body is free during flight and its angular velocity is constant[42]. This last version of the controller has not been implemented yet, because the low gravity setup does not allow for such an assumption as we will see in § 6.1.

5.3 Actual implementation

5.3.1 Leg control

The leg is quite well positioned with a linear controller. The sum of the control signals sent to the servos determines the pitch angle of the leg, $\beta_p = k_p \cdot (\text{servo1} + \text{servo2})$, and the difference is responsible for the roll angle, $\beta_r = k_r \cdot (\text{servo1} - \text{servo2})$. As k_r and k_p are not a power of 2, lookup tables[43] have been implemented to translate the leg angles into servo signals.

Moreover, the bow string and the thrust mechanism also affects the pitch leg angle (see § 2.3.2). But, for a first approach, we can make the control for the retracted state[44] of the leg and add an offset to the servos during stance. This offset, called *servo compensation*, should lead to a vertical position of the leg when not retracted. Some problems occur during stance, which will be discussed in § 6.5. Fig. 5.3 shows the synchronization between the state of the robot, the thrust mechanism and the servo compensation. The black lines are the logical signals or the commands and the dotted gray lines represent the real position of the actuators.

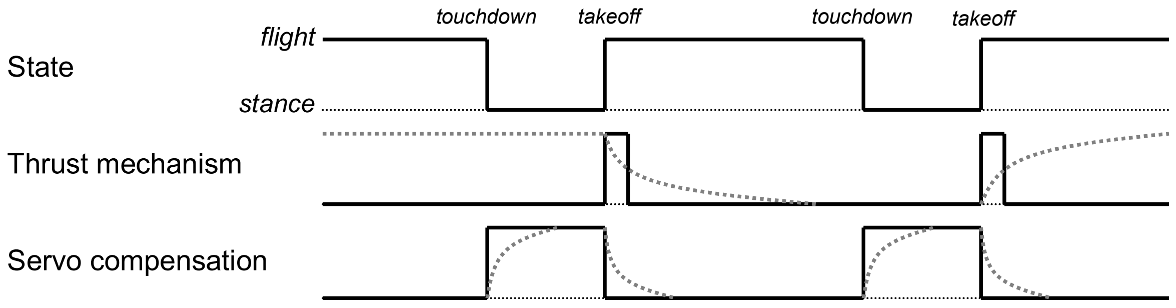


Fig. 5.3 – Servo compensation and thrust actuation

Notice that the leg is controlled in open loop. For the moment, there is no real feedback from the position of the leg in order to compute the error between the requested angles and the real ones and thus no possibility to implement a position controller. However, a sort of control is present inside the servomotors themselves.

5.3.2 Finite state machine

A state machine tracks the hopping behavior to synchronize the actions of the controller. A simple state machine, based only on the foot sensor information, may be sufficient for the very first tests (see Fig. 5.3).

If some more complex controllers are implemented, the state machine must be refined. Fig. 5.4 is an example of such a state machine.

[42] This is not the case when using the first trial setup for low gravity, see § 6.1.
[43] `Req2BetaR` and `Req2BetaP`.
[44] The leg position only truly matters at touchdown. Therefore, some leg motion during flight is acceptable.

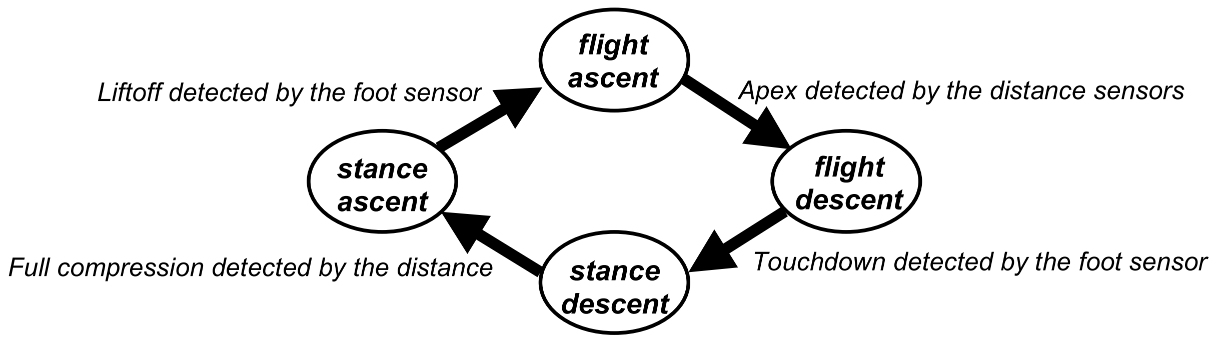

flight ascent
Liftoff detected by the foot sensor
Apex detected by the distance sensors
stance ascent
flight descent
Full compression detected by the distance
stance descent
Touchdown detected by the foot sensor

Fig. 5.4 – State machine

Of course, even more precision may be required. For instance, see [Rai86], p. 41, for the presentation of the state machine used on its first planar hopper. The one employed on the 2D Bow Leg Hopper is explained in § 4.3.1, in [Zeg99].

5.3.3 Structure of the controller program

The main program is divided into several loopS based on a real time counter. Every millisecond, the main loop is executed, which contains the other ones that have their own millisecond counter in order to be executed at the appropriate rate.

Below is a quick description[45] of what is accomplished at which frequency.

- Every millisecond
    - Update of the state machine, based on the foot sensor (see § 5.3.2)
    - Pulse the retract mechanism solenoid if needed
    - Reading of the AD converter (4 channels for the 4 Sharp sensors)
    - Clustering function for the sensors (as described in § 4.4)
- Every 5ms
    - Initialization of the pulses for the gyros
    - Reading of the 3 radio receiver channels
    - Update of the servos output
- Every 10ms (previously 5ms)[46]
    - Servo PWM pulse initialization
- Every 40ms
    - Body attitude and COM height computation (as described in § 4.5)
    - Estimation of current attitude by taking care of the signal from the gyros
    - Update of the state machine, if required (see § 5.3.2)

Then, a single interrupt routine, executed every 4.34 microseconds is responsible for:

- Updating the real time counter
- Ending the pulses for the gyros and the servos
- Reading the length of the pulses coming from the radio receiver
- Reading the length of the pulses coming back from the gyros
- Sending each bit for the UART routine

[45] Not shown is the transmission of data to the BLHSupervisor.
[46] The update rate of 200Hz was too high for these servos, see § 3.1.3.

6 Initial experiments

Preliminary laboratory experiments have shown that the robot is capable of performing some set of jumps, under low gravity – about half the normal gravity – driving by a human using the radio command (see § 6.5). The jumping tests have been made with and without any control.

The most important, in a first phase, is to determine certain parameters like the body swing frequency or the leg oscillation in order to improve the mechanical design and the controller.

6.1 Test setup

In order to prevent the robot from damage during jumping tests, a security string has been employed that link the body to a point in a high ceiling. This string is long enough in order not to perturb the robot but short enough to hold it just before it hits the floor.

Further, a long rubber band, in parallel to this security string apply a force to the robot that is about half the gravity. This system has to effects:

➢ **Low gravity**

First it slows down the hopping frequency by decreasing the actual gravity applied to the body. From Eq. 5.3, we can deduce:

$$T_f = k\sqrt{\Delta h} \quad \text{where} \quad k = \frac{2\sqrt{2}}{\sqrt{g}} \cong 0.9 \qquad \textbf{\textit{Eq. 6.1}}$$

Δh [mm]	50	100	150	200	250	300
T_f [ms]	200	285	349	402	450	493

Table 6.1 – Time of flight under normal gravity

With half the normal gravity, k=1.28:

Δh [mm]	50	100	150	200	250	300
T_f [ms]	280	405	496	572	640	701

Table 6.2 – Time of flight under half the normal gravity

Increasing the period of jumps allows more time to store energy in the leg – which was necessary with the first thrust mechanism – and also more time to get the leg quiet after takeoff.

➢ **Stabilizing the body**

Because the hook where the rubber bands are attached is well above the COM of the body (see Fig. 6.1), a moment is applied to the body.

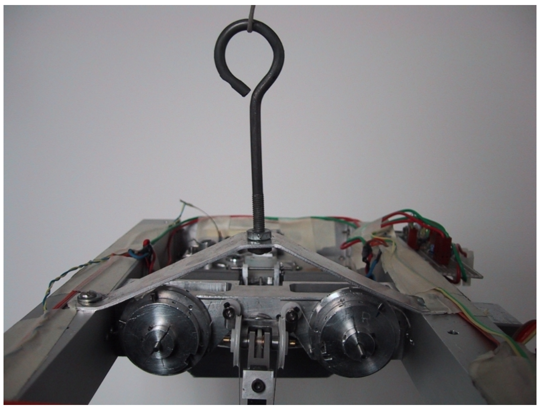


Fig. 6.1 – The hook above the body COM that is the attach point of the rubber band

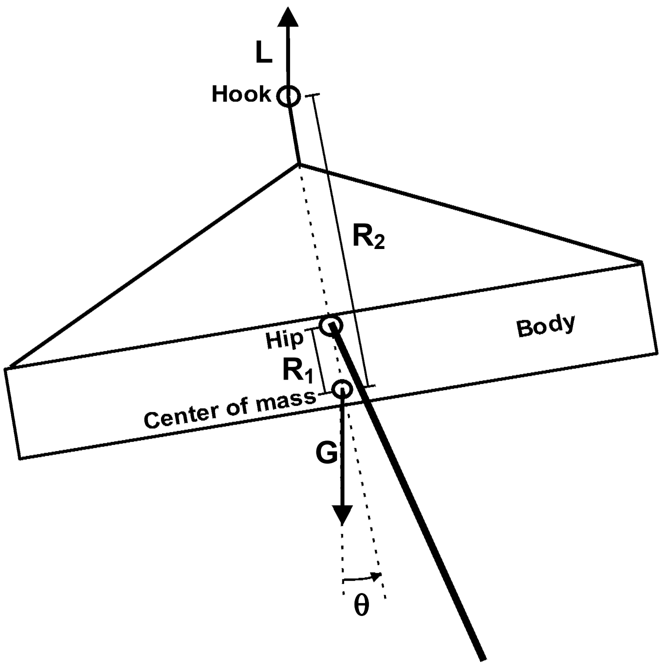


Fig. 6.2 – Forces on the body during flight with the low gravity system

The torque at the COM during flight can be expressed by:

$$\tau_f(t) = L \cdot R_2 \sin(\theta(t)) \qquad \textit{Eq. 6.2}$$

This moment tends to keep the body horizontal.

As said before, the body of the Bow Leg Hopper should be passively stabilized (see § 5.1.3). The planar hopper has successfully demonstrated this effect. The concern is that we don't know the importance of the damping moment, which was applied by the boom to the body through the bearing joint. With this stabilizing effect of the low gravity system, this crucial point has not been really addressed on the 3D machine, yet.

6.2 Determination of the body swing frequency

However, in order to prepare for the removing of the low gravity system or simply attaching to a lower point on the robot, near the body COM, the swing frequencies of the body have been experimentally approximated. The following graphs (Fig. 6.3 and Fig. 6.4) show the gyro output when the body is simply swinging around one of its main axis (roll and pitch), attached at the hip location using the balancing setup described in § 2.3.3.

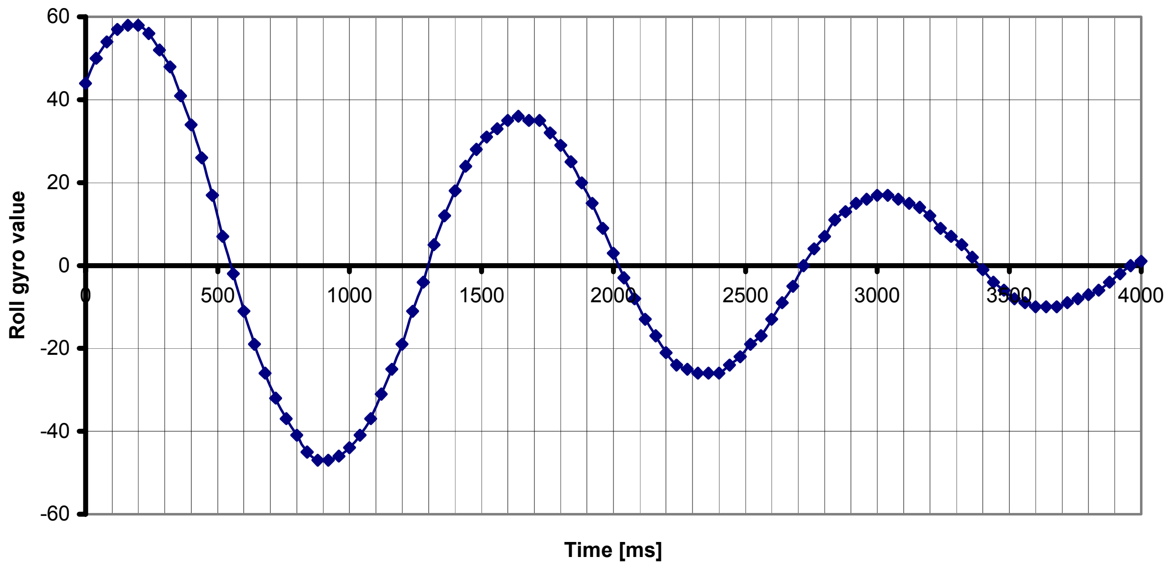


Fig. 6.3 – Oscillation around the roll axis

This graph shows the roll gyro value that is proportional to the roll angular velocity. The oscillation frequency can be deduced the period (1440ms) of the oscillation:

$$f_r=0.69\ [Hz] \Rightarrow \omega_r=4.36\ [rad/s]$$

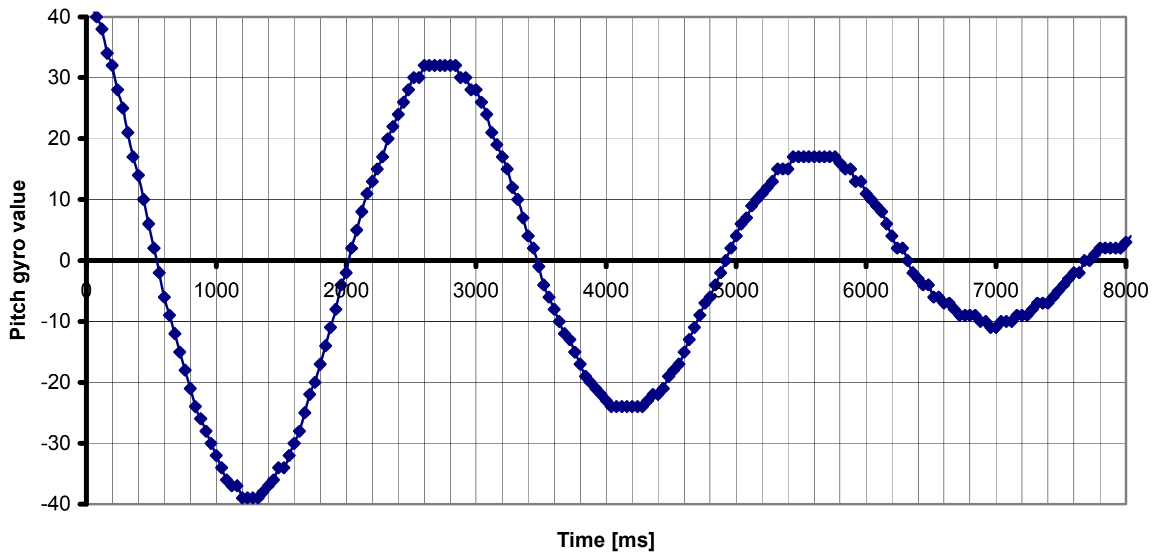


Fig. 6.4 - Oscillation around the pitch axis

This second graph permits to determine the pitch swing period (2880ms) and leads to:

$$f_p=0.35\ [Hz] \Rightarrow \omega_p=2.181\ [rad/s]$$

Of course, because the body is less long in the pitch axis direction, the roll frequency is higher thus the more critical. This period (1440ms) is larger[47] than the normal flight time displayed by

[47] Two to six times.

the robot, even under low gravity. Notice that the pulsations calculated here are not exactly the same as the one of Eq. 5.10. Actually, here we are calculating:

$$\omega = \sqrt{\frac{MgR_1}{J + MR_1^2}} \qquad \textbf{\textit{Eq. 6.3}}$$

But the effect of the additional term deduced from the Steiner theorem is negligible[48], since we are interested by order of magnitude. Actually, we don't know if this frequency is low enough or not. Some experiments will follow to determine this and take the appropriate measures.

The above analysis suggests that we may want to augment the body inertia to slow down the pendulum frequency and make the COM location less critical.

6.3 Keep the leg vertical

Since the new thrust mechanism was ready only two weeks before the due date, the first controller described in § 5.2.2 has been implemented and tested in a non-jumping case. It is quite impossible to measure the accuracy of such a controller that keeps the leg vertical when the body attitude changes dynamically. In order to demonstrate the results, some videos have been made that are available on [www3DbowLeg]. They shows the difference of the control with and without the gyro compensation. Before the hand sign, the gyro compensation is off. After the sign it is turned on. The results are quite good, above all if the COM height is not too important.

6.4 First thrust mechanism

As described in § 2.3.4, a first retract mechanism based on a hobby servomotor was first mounted on the 3D Bow Leg Hopper. This experiment demonstrates that this motor was too weak. The signal from the potentiometer inside this servo has been recorded during one or two jumps under low gravity (about half normal gravity). Fig. 6.5 shows the height of the body COM, the position of the retract servo and its command (dotted lines).

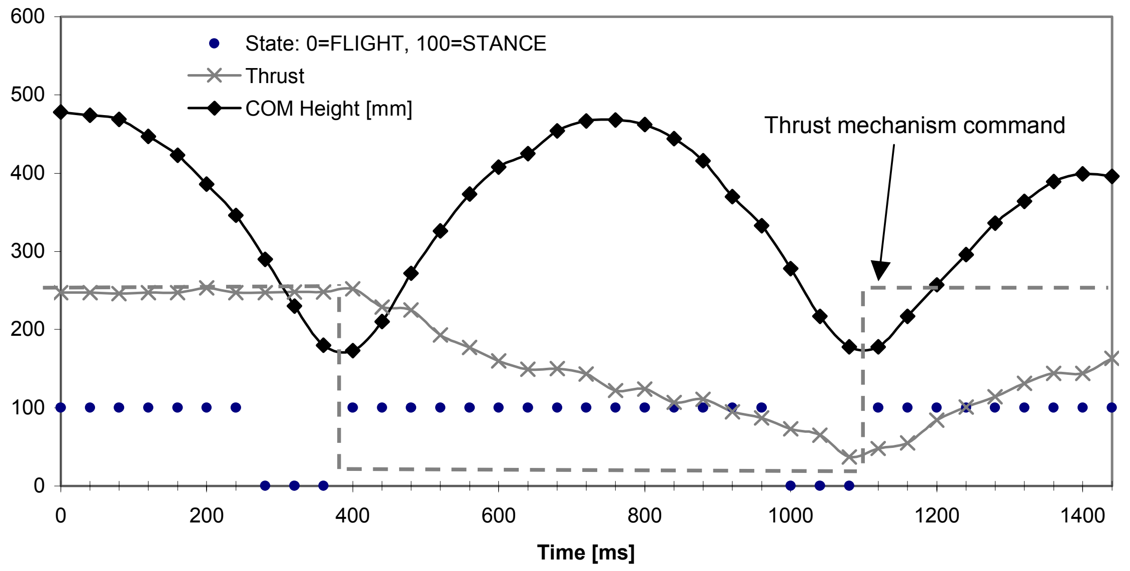


Fig. 6.5 – The first thrust mechanism, based on a standard servo was too slow

[48] A quick analysis shows that the radius of gyration contained in J is much more larger than R_1.

Before the jump in the center of the graph, the thrust servo was quiet at one of its initial positions. Just after takeoff (see the *State* indication), it begins to move. Although the following bounce lasts quite long (600ms), we can see that the retract servo does not reach the opposite position before the next touchdown. More over, the curve is not a line and shows some hesitations certainly due to the servo overload.

Another interesting thing to look at on this graph is the latency in the computed heights. Actually, the minimums of the COM height curve should be located at the middle of the stances. Here they are about 100ms later which correspond to the phenomena described in § 4.5.

6.5 First jumps of the 3D Bow Leg Hopper

Some very first jumping experiments have been realized under half the normal gravity. Most of them were simply filmed. A good analysis may be performed afterwards by watching the robot movements frame by frame.

The main problem that appears so far, after the changing to the new thrust mechanism that seems very accurate and fast, is the leg oscillation after liftoff that may last until the next ground contact. Here is a list of parameters that could help to reduce them:

- Introducing a delay between the thrust mechanism actuation and the servo compensation after takeoff (see Fig. 5.3). Another way to better synchronize the compensation would be to implement a ramp or to really measure the bow string position with a linear potentiometer.
- Slow down the thrust motor.
- Measure the real leg or foot position and implement a better feedback control. This control could take care of the real position of the leg at liftoff and adapt the control servos to this position before trying the reach another one.
- Reducing the tension in the rubber bands that are mounted in parallel to the control strings.
- Increasing the radius of the bow string pulleys and pitch pulleys.
- Decreasing the moment of inertia of the leg, thus essentially the mass of the leg.
- Introducing a damping moment in the hip. Care must be taken not to introduce important torques between the leg and the body during stance.

The following graph (Fig. 6.6, recorded with the BLHSupervisor on February 15, 2001) shows a good set of successive bounces under low gravity. The robot was driven via the radio command and jumping on a carpeted floor.

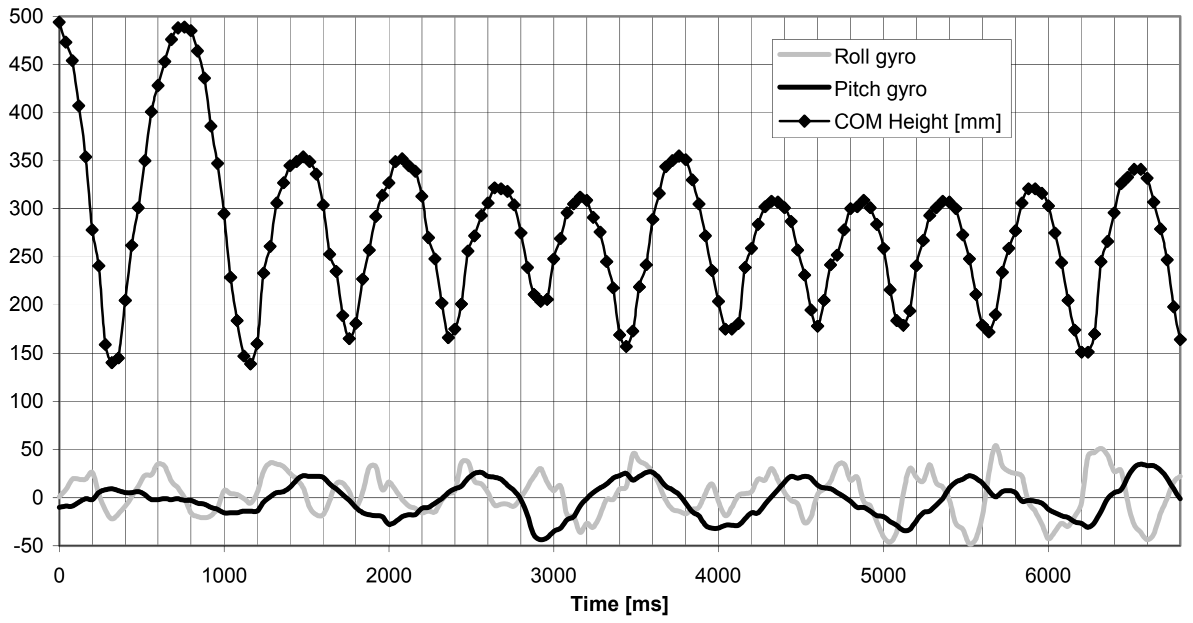


Fig. 6.6 – Eleven jumps under half the normal gravity and driven via the radio command

Some interesting things are visible on this graph. First the average time of flight on these 11 bounces is about 595ms without taking care of the stance time, which has been measured at about 130µs in the same conditions. Thus we have a mean flight time of 465ms. The average Δh is about 140mm. These values correspond to the ones shown in Table 6.2. This means that the settings[49] of the low gravity system are quite good.

Then, the gyros illustrate the impact of the low gravity system on the body behavior during flight. The body angle velocities are clearly not constant and the oscillations around the roll axis are quite twice as fast as the ones around the pitch axis because of the difference between J_r and J_p as explained in § 5.1.3.

The situation is quite delicate because the low gravity system introduces some oscillations of the body during flight when it would be helpful to have a constant angular velocity, for instance for the third controller described under § 5.2.2. But, on the other hand, the stabilizing effects allow for testing the mechanics without taking care of the passive body stabilization.

[49] It is made by simply weighting the robot with and without the rubber band.

7 Conclusion

7.1 Future work

The following discussion about future work is split into the short-term improvements and tests, and the general continuation of the Bow Leg Hopper project.

- **In the short term:**
  - As described in the previous chapter, the leg oscillations after takeoff are quite important. Thus, the most urgent improvement to do is trying to reduce these oscillations by some damping moments. This involves mechanical changes or tricks.
  - Another possibility for reducing these leg vibrations is to adjust the timing of the servo compensation related to the thrust mechanism actuation. Also, the measurement of the bow string length with a linear potentiometer may allow a better control of this compensation.
  - The gyros have to be calibrated in order to take a larger place in the body attitude estimation (see § 5.2.2).
  - The passive body attitude concept has to be assessed by hanging the low gravity rubber bands nearer to the COM of the body or by accomplishing some free jumping experiments.

- **In general:**
  - The most important future work is the development of a system for measuring the leg position in real time. This system should permit to implement a much better control of the leg placement and lead to dramatically reduce to leg oscillations. A preliminary idea is to equip the foot with an LED and measure its location with a PSD attached right bellow the hip, on the body. This system will also allow determining lateral velocity of the machine based on the foot during stance.
  - A new body attitude sensing system could be designed, involving more dynamic sensors. Developing dedicated range finders like wide-angle sonar transducers (or acoustic lens) or based on the laser-triangulation concept (with a PDS) may lead to dramatically decrease the latency and improve the precision at high hopping height.
  - Real gyroscopes – like the ones that equipped the Raibert 3D monopod – may be useful to compensate for the high latency in the body attitude computation.
  - A radio for replacing the current RS232 interface would be a very useful tool for recording data during future free jumping trials.
  - Finally, a re-design of the mechanics may be required in order to help the passive body stability. An active lateral hip positioning or a gyrostabilizer represent some of the preliminary ideas.

7.2 To conclude…

Working four months on such a complex and utopian project doesn't allow accomplishing very impressive progress. Many people have contributed to the field and will continue to develop running machines. The way is long and sometimes laborious. Remember that, so far, most robots move quasi-statically, and most dynamic robots either fly (with smooth inertias), have continual ground contact (fast walking), have intrinsic orientation stability (rolling), or a kinematic connection to ground (manipulators).

But now imagine yourself, walking in a corridor and meeting a little jumping monopod turning toward the stairs and disappearing by climbing up the steps. This only thought is so exciting and surrealist that it is enough to push a number of scientists toward this futurist vision.

The 3D Bow Leg Hopper represents a significant advance in the domain, being the first self-contained one-legged robot. It is in its early youth and just accomplished its first steps, assisted by some external support not unlike children helped by their parents. But the concept exists and certainly has a fine future ahead of it.

Designing the body orientation sensing system and a simple controller for the very first jumping tests represent my main contribution to this project. A new milestone has been reached and the next step will be to sense the body lateral velocity in order to implement a real controller and give more autonomy to the robot.

7.3 Artistic insight of the 3D Bow Leg Hopper

Lune creuse
Fragile nudité d'arche
Vide
Lune sans reflet
Sous des eaux assoupies
- Où s'anéantira cette lueur fugitive ?

Quelques astres y rêvent
Parmi silence et fureur
Tels Chiffres, Sciences et Formes
Fulgurent de leur découverte.

Serait-ce là
Absente et perdue
De nul écho poursuivie
Surgie d'un obscur désastre
Avec la densité du mystère
La parole tout bas
D'une secrète étoile ?

Car
Voici que palpite l'espace
Flamme devenue
Par l'ignition du mouvement
Tout à déployer
Un vol ivre
Qui vers l'azur délire.

Embrasement d'aile
A jaillir d'une lourdeur d'écume
Parmi l'absence
À briser d'une lourdeur d'écume
L'absence.

Une fraîcheur de flamme
T'enlace à chaque battement
Dont le bond captif
Recule l'horizon
Qu'il se nie ou s'accroisse
Impassiblement.

Equilibre :
Ô Vertige
Tel qu'en lui seul
Le mouvement le change
Avec pour langage
Rien que le scintillement
Au ciel
Infiniment
D'un rythme
D'incandescence.

Julien

7.4 Acknowledgements

I would especially like to thank Ben Brown and Dr. Garth Zeglin for their assistance, help and patience during my whole stay in The Robotics Institute.

I am equally indebted to Prof. Illah Nourbakhsh and Prof. Roland Siegwart who made my stay in the USA possible and provided pertinent advices along the way.

Thanks to the artists, Julien Zufferey for the little text in French about the 3D Bow Leg Hopper and Fran6 Ray for the humoristic drawing on the cover.

My officemate often provided useful comments and much patient listening. Thank you Adrian Hilti. I am also grateful to Jean Harpley for her assistance.

My work was supported by Heinz Endowments.

Pittsburgh, February 21, 2001

Jean-Christophe Zufferey

8 Appendix

8.1 Symbols and notations

8.1.1 Frames

- The world frame is called *W* and the z-axis is perpendicular to the floor, in upward direction.
- As the global position of the robot and its movement around the yaw axis are not under the scope of this project, a reference coordinate system (named *R*) has been chosen such as its center is at the location of the hip, the x and y axis define a plane parallel to the floor (x is the roll axis and y the pitch axis) and z-axis is always vertically oriented.
- A second frame called *B* is linked to the body of the Hopper. Its origin is the same as *R*. x- and y-axis define the body plane and z is perpendicular to this plane.

8.1.2 Suffixes

r	roll
p	pitch
t	touchdown
l	liftoff
a	apex
f	flight
s	stance

8.1.3 Physical constants and machine parameters

M	body mass
g	gravitational acceleration
F_0	force of a constant force leg spring

8.1.4 Geometric parameters

l	leg length
h	height of the COM wrt *W* (*note: in some occasion, the COM and the hip are taken as co-incident, for sake of clarity)*
θ_r, θ_p	body roll (around x-axis) and pitch (around y-axis) angles wrt *R*
ϕ_r, ϕ_p	leg angles wrt *R*
β_r, β_p	leg angles wrt *B* (actually, $\beta = \phi - \theta$)
R_1	distance between hip and COM
R_2	distance between hook and COM

8.1.5 Kinematics

Δh	vertical distance the hopper falls from apex to impact
Δl	maximum leg compression during stance
t	time
n	index for the number of jump (a jump begins and ends at touchdown)
τ_s	torque during stance
τ_f	torque during stance (with the low gravity system)
f	body swing frequency [Hz]
ω	body swing pulsation [rad/s] ($\omega=2\pi f$)
J	body moment of inertia around one of the principle axis (roll or pitch)
ξ	energy stored in the leg

8.1.6 Sensors

s	distance (span) between two opposite sensors
i	sensor index (i=1..4)
d_i	straight distance from the sensor (back of the packaging) to the floor
h_i	height above floor of sensor i

8.1.7 Time

t_t	touchdown time (moment of impact)
t_a	apex time
t_l	liftoff time
T_s	stance duration
T_f	time of flight
T_{fall}	time from apex to ground contact (touchdown)
T_r	time from liftoff to apex

8.2 Abbreviations

AD	Analog to Digital
COM	Center Of Mass
DOF	Degrees Of Freedom
FSR	force-sensing resistor
LCD	Liquid Crystal Display
LED	Light Emitting Diode
PCB	Printed Circuit Board
PSD	Position Sensing Device
PWM	Pulses With Modulation
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver Transmitter (often used to designate a RS232 serial interface)
VP	Virtual Peripheral
wrt	with respect to (e.g. wrt world coordinates)

8.3 Bibliography

[Ale90] R. McN. Alexander. **Three Uses for Springs in Legged Locomotion**. IJRR, vol. 9, n. 2, 1990.

[BZ98] H.B. Brown and G.J. Zeglin. **The Bow Leg Hopping Robot**. In Proceedings of IEEE International Conference on Robotics an Automation, 1998.

[McM84] T.A. McMahon. **Mechanics of Locomotion**. IJRR, vol. 3, n. 2, 1984.

[Rai86] M. H. Raibert. **Legged Robots That Balance**. The MIT Press, 1986.

[ZB98] G.J. Zeglin and H.B. Brown. **Control of a Bow Leg Hopping Robot**. In Proceedings of IEEE International Conference on Robotics an Automation, 1998.

[Zeg99] G.J. Zeglin. **The Bow Leg Hopping Robot**. PhD Thesis, CMU Robotics Institute, 1999.

8.4 Related web sites

[www3DBowLeg]	dmtwww.epfl.ch/~jzuffere/BowLegHopper
[www2DBowLeg]	www.cs.cmu.edu/~garthz/research/bowleg
[wwwjczSensors]	dmtwww.epfl.ch/~jzuffere/RangeFindersComp_E.html

[wwwLegLab]	www.ai.mit.edu/projects/leglab
[wwwSandia]	www.sandia.gov/media/NewsRel/NR2000/hoppers.htm
[wwwUbicom]	www.ubicom.com
[wwwByteCraft]	www.bytecraft.com
[wwwSonaSwitch]	www.sonaswitch.com/SonaSwitch/mini.asp
[wwwSharp]	www.sharp.co.jp/ecg/db19992000/contents/osu_f.html

8.5 Electronic board schematic