Design and Stabilization of a One Legged Hopping Robot

B.Tech. Project

of

Pratik Chaudhari

Roll No : 06D01015

under the guidance of

Prof. Hemendra Arya
Department of Aerospace Engineering, IIT Bombay.

and

Prof. Bhartendu Seth
Department of Mechanical Engineering, IIT Bombay.

Indian Institute of Technology Bombay

April 15, 2010

Certificate

This is to certify that this report of **Pratik Chaudhari** on the topic, **Design and Stabilization of a One Legged Hopping Robot** towards fulfilment of the requirements of **B.Tech. Project AE 497** is approved by me for submission. It represents the work carried out by the student under my guidance. Prof. Bhartendu Seth was the co-guide for the first part of the project.

Prof. Hemendra Arya

Guide

April 15, 2010

Abstract

Single-legged locomotion gait is a hopping motion consisting of alternate flight and stance phases. In such a hopping robot, if the energy lost in friction and impacts is compensated, then control of robot attitude can result in stable hopping motion. In this regard, use of a reaction wheel mechanism as an attitude control actuator is a novel and energy efficient approach.

This report discusses the development of the mechanical and electronic systems of a one legged hopping robot. An energy efficient mechanism has been fabricated for experiments to demonstrate a stable running gait as well as in-place hopping using SLOM concept. A test-rig was also built to pivot the robot near its C.G. to perform attitude orientation experiments independently of hopping. Electronics developed during the course of this project consists of an onboard controller, DC motor drivers using MOSFETs and an inertial measurement unit.

Dynamics of impulsive systems is significantly different due to its hybrid nature and hence a detailed model of the hopping robot was formulated to study it. This involved simulations for a running gait, in-place hopping as well as exercises in finding the stability basin of the hopper design. Controllabilty of non-linear systems and optimal control techniques were also studied to be implemented later.

A full-state attitude feedback system was developed using inertial sensors since the hopping system does not have physical contact with any fixed reference during the flight phase. A kalman filter was designed to attain drift free orientation feedback and was implemented in a real-time embedded controller.

Keywords: SLOM, offset-mass, reaction wheel, hopping robot, non-linear hopping gait model, inplace hopping, poincare map, attitude control, PID, kalman filter, inertial sensors.

Contents

List of Figures

Chapter 1

Introduction

The motivation for research in legged robotics has been to understand human motion and legged motion in general. There are various applications that spring to mind when one thinks of the uses of legged robots viz. travelling on difficult terrain, search and rescue operations in event of fires and landslides, space exploration etc. At the same time it fulfils the science fiction dream of having a running and jumping robotic pet! Some of the major challenges in development of legged robots are, [1]

1. Stronger energy pumping mechanisms are needed to compensate the energy loss after impact with the ground. Heavier the robot, more energy is lost every impact which results in larger and even heavier actuators to compensate it.

2. Dynamics of legged robots is significantly more complex than wheeled ones. Control strategies employed for different actions like hopping and running are much different from conventional ones.

3. Energy efficiency is a major concern due to multiplying effect of any extra weight added to the robot.

1.1 Previous Approaches

SLIP

Marc Raibert pioneered the field of legged robotics [2, 3]. He developed three pneumatically actuated one legged robots to demonstrate 3D hopping. There have been a variety of approaches towards building better actuators and stabilization strategies. We shall look at them briefly in the following sections. Sayyad, Seth and Seshu review the development of one legged robots in detail in this review paper [1].

1. **Energy Pumping Mechanism (EPM)**
   Pneumatic actuators were seen in Raibert's *Monopod* [4] and Zeglin's *Uniroo* [5]. It was observed that electro-mechanical actuators were much more efficient than pneumatic or

hydraulic actuators. ARL Monopods developed by Buehler [6, 7] that utilized a ball-screw mechanism to store energy in a leg spring and had significantly better energy characteristics. Zeglin's planar bow-legged hopper used a flexible bow shaped leg positioned using servomotors [8]. Almost all approaches after this have used springs to store energy and provide impact forces for liftoff; only major differences being whether they used springs in a telescopic leg or as a part of the joints in an articulate leg.

2. **Stability**
   Raibert's hoppers used a pneumatically controlled actively balanced hip [1, 6]. There was another approach wherein swinging arms were used as passive stabilizers. Note that such appendages do not provide a complete solution to the problem but just reduce the energy required for balancing. ARL Monopod II had a compliant hip-leg using a swinging leg [9, 7].

The above robot have impact forces passing through their C.G. to provide a stable system for single place hopping and are referred to as *Springy Leg Inverted Pendulum* (SLIP). It is noted that SLIP does not account for the pitch stabilization problem which is of practical concern.

SLOM

Shanmuganathan et. al. considered asymmetric configurations in which the CG location was offset from the geometric center. This is referred to as a *Springy-Legged Offset-Mass* (SLOM) hopper [10]. The impulsive torque acting during the stance gives a pitch up velocity in the flight phase. This compensates the net pitch down during the stance phase due to the horizontal velocity. Sayyad and Seth have analyzed this configuration using a 3D Poincaré map to obtain a periodic motion stabilized by observer based state feedback strategy [11].

1.2 Stage I

The SLOM hopper was used as a test bed for devising hopping strategies by Saboo [12], Simit [13] and Siraj [14]. However, it was an over-designed system with large energy losses due to impacts and friction. It was necessary to remove these flaws in the robot before further work could be done on it. Hence, it was decided to go ahead with a completely new mechanical design for the hopper keeping the following things in mind about the previous design.

1. Compression springs need an enclosure outside them to keep them in place when in compression. This results in frictional losses in every cycle of energy pumping. Tension springs on the other hand do not have such frictional losses associated with them.

2. The SLOM hopper could not hop above a height of 10 cm. The energy pumping mechanism (EPM) was the limiting factor. For achieving heights larger than this, we need to significantly reduce the leg mass and have a more energy efficient EPM.

3. A reaction wheel should be present on the hopper and this will be used to reorient the robot to demonstrate both in-place hopping and running capabilities.

Work done

1. Two mechanical designs that we formulated to try to obtain an efficient energy pumping mechanism for the one legged hopper robot. Both the designs had their share of flaws and it was not possible to go ahead with the fabrication without further analysis on any one of them.

2. The sizing analysis for the various generic components of the hopper was done to arrive at approximate values of the rack and pinion dimensions, motor torques, electronic considerations, mehanical dimensions and forces.

3. The next step was to re-run this analysis side by side with the fabrication unit to make a feasible design that can be manufactured easily as well as serves our purpose.

4. The electronics for the hopper was re–designed. Older circuits can be used for preliminary experiments. Newer self-made motor drivers using MOSFETs were made to ensure a good enough drive to the motors. The IMU will be same as the one used by Siraj and Simit [14, 13]. We think that just the pitch attitude is sufficient provided we constrain the robot properly in the other two axes. The micro-controller unit has been redesigned as the new system has 2 DC motors, 2 servo motors and an IMU.

1.3 Stage II

The goals of the second stage of the project were two-fold,

Implementation

1. Fabricate the robot, choose materials, feasible dimensions etc. The sizing analysis done in the first stage proved particularly helpful for re–analysing new designs.

2. Design and fabricate electronic circuits for onboard computation and control.

3. Implement the above controller onboard the hopper and demonstrate a stable running gait.

Theoretical

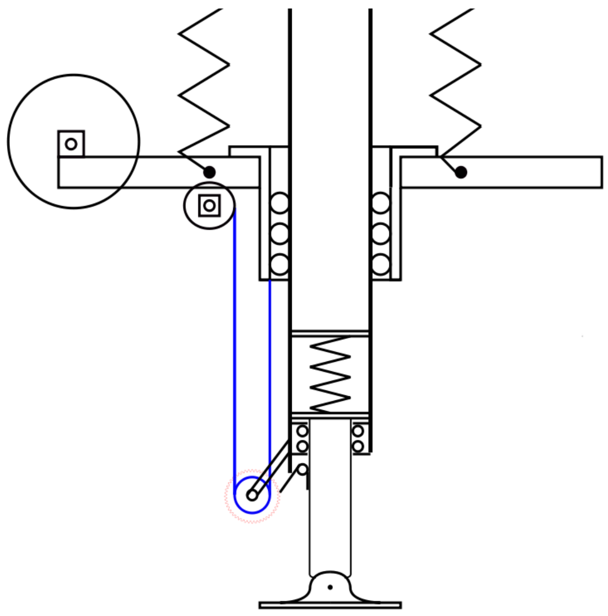
1. Model the hopper system and study dynamics of impulsive systems in general.

2. Devise a control strategy to pump in energy as well as re–orient the hopper in mid-flight. Simulate the dynamics and control strategies to find out the stability basin of our design.

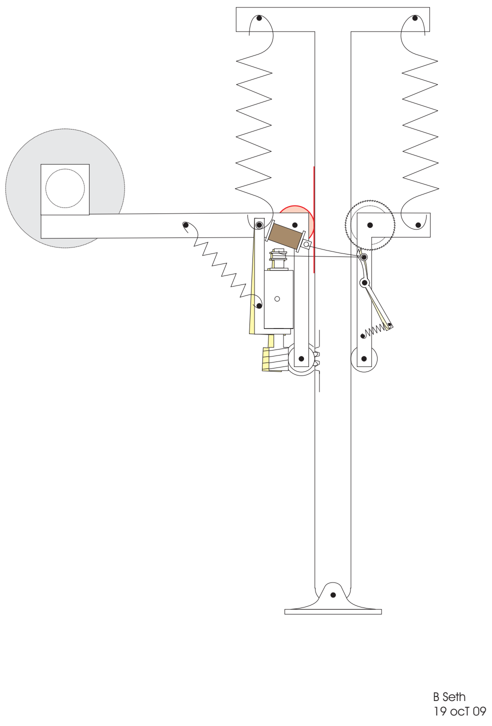
Chapter 2

Robot Design

The final design of the hopping robot is described in this chapter. We first look at two previous designs devised to make use of SLOM effect. After ironing out some deficiencies, we arrive at a simplistic design that satisfies our criteria.



(a) Design 1



(b) Design 2

Figure 2.1: Previous hopping robot designs : Approx. height = 500 mm

2.1 Mechanical

Previous Designs

A description of the two designs is given in the report for Stage I [21]. The evaluation of these two designs is as follows,

Design 1

1. It has to be ensured that the winding winch does not slip over the pulley when the platform is suddenly released from the constraint. At the same time, the winch must be free enough so as not to hinder the movement of the platform after the release.

2. The platform moves over the leg with the help of a circular bushing. There is another bushing for the lower leg to move into the upper leg. These circular bushings are light and durable.

3. The hatch that is a crucial part of the energy release and the constraint mechanism is the weakest part, but it can break easily.

Design 2

1. The sleeve of the motor is kept on the same side as that of the reaction wheel and helps to provide an offset mass for the SLOM effect. Body mass also helps to obtain an offset C.G.; thus the distance of the reaction wheel from the axis need not as be as large as calculated in the adjoining analysis.

2. The worm-worm wheel on the rack mechanism provides a huge mechanical advantage and thus reduces the maximum torque required from the motor. This scales down the mechanical system as well as the electronic system requirements.

3. All the force of the extension springs is coming as an axial load on the shaft of the motor. We thus need to choose a gearbox that can handle these axial loads.

4. The friction pulley has to work against the pual spring, sleeve spring and the horizontal component of the rack force ($k\,x\,tan\,\theta$) to keep the string in tension. This is compounded by the fact that the there is a maximum ω the motor can accelerate to in the energy storing phase. It is much better if this ω is dictated by the torque requirements which are as critical rather than this mechanism.

5. Less resolution on the desirable extension of springs because we are operating on a rack. This can however be easily taken care of in the pitch control law.

6. The constraint mechanism for Design 2 is not reliable enough and should be improved upon. The major problem is to provide an opposing force to the horizontal component of the rack–worm-wheel force ($k\ x\ tan\ \theta$). This force has to be present when the motor

is extending the springs and should be removed before we release the ratchet so that the platform along with the motor-sleeve is free to move up. To disengage the motor from the rack, the spring shown in the left part of Fig. 2.1b will have to be there as well.

Final solution

Both the designs given in Section 2.1 had significant drawbacks and hence a newer design was made utilizing their individual mechanisms and the knowledge gained from analyses.
The salient features of this design are as follows,

1. As shown, in the figure, it consists of two large flanges with all the components arranged inside it. Enclosing the components inside the flanges makes it a compact, closed system with only essential components like batteries, motors, circuit boards being accessible from outside.

2. One major component of this design is the rack and pinion mechanism. It is difficult to cut a 45 degree rack due to fabrication constraints. However, this mechanism has been built and fixed on the robot successfully.

3. The whole robot is built in such a way that it can be pivoted fully near its C.G. to a test–rig. This test–rig simply consists of two large columns from which the robot hangs. It is a simple contraption and should prove handy while performing experiments on the robot.

4. The ratchet and paul mechanism is on the same axis as the band drive. There is a servo motor on the outer side near the paul to actuate it (which is not shown here).

5. Another servo motor sits on the vertical portion of the robot to ensure that the motor sleeve gets detached after extending the spring.

6. We have ensured that the C.G. of the system lies along its centerline and hence there are no lateral moments. This is done by putting batteries on the opposite side of the reaction wheel motor.

2.2 Analyses

We need to find approximate design estimates of the hopping robot by performing analysis. The following three values are what we are interested in.

1. Masses of the platform and the leg

2. Dimensions of the reaction wheel based on the above masses

3. Choice of winding and reaction wheel motors

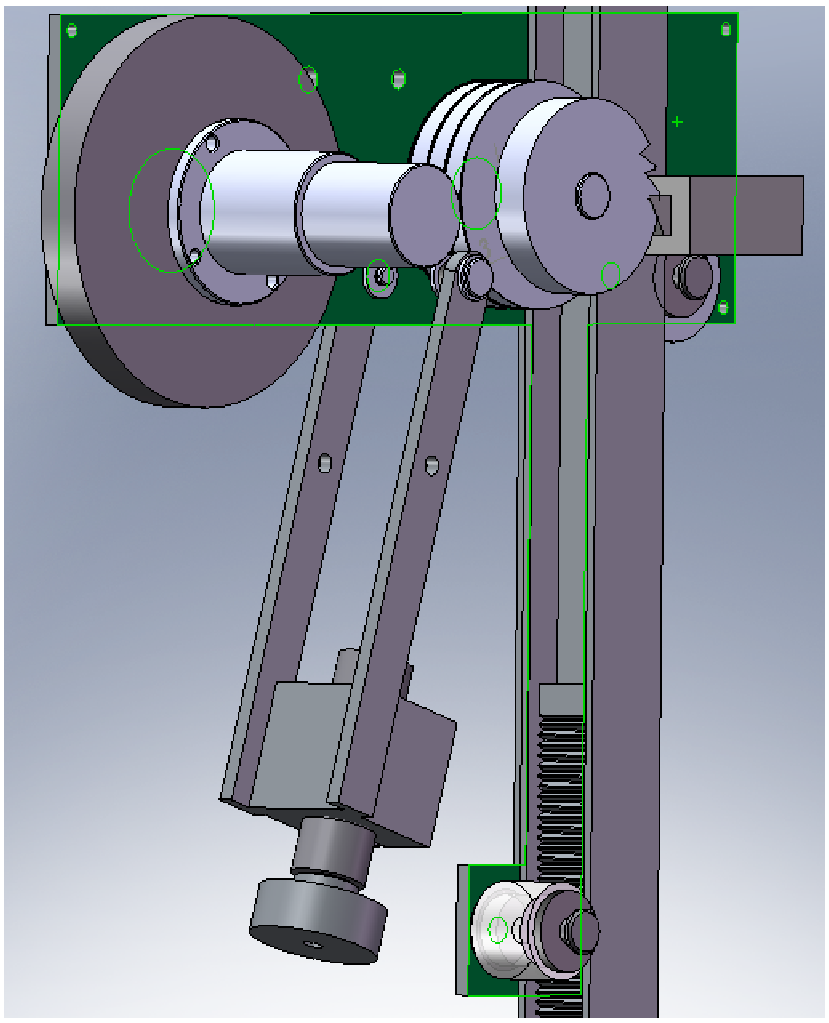


Figure 2.2: Solidworks hopping robot assembly : Motor sleeve, reaction wheel and rack and pinion. Note that one of flanges is hidden in the picture to see the internal components

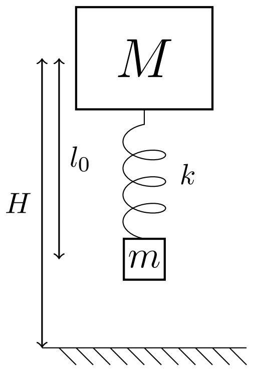


Figure 2.3: 2 mass problem [13]

2 mass problem

The basic idea behind a hopper is like that of the 2 masses connected by a spring problem. If the system shown in Fig. 2.3 is allowed to fall from a height, the heavier mass pulls the smaller mass with it back into the air after impact. Every cycle is accompanied by a loss in energy due to the inelastic impact of the smaller mass with the ground. If we pump this energy back into the system using an external agent in every cycle, we can ensure sustained hopping at the chosen height. The 2 mass problem can thus be taken as a basis to compute the range of values of masses for acceptable performance. It is seen from Fig. 2.3 that if h_i are progressive heights, we have the relation,

$$h_n = \frac{Mh_{n-1} + ml_0}{M + m} \tag{2.1}$$

$$E_{loss} = \frac{Mg\,(H - l_0)}{1 + M/m} \tag{2.2}$$

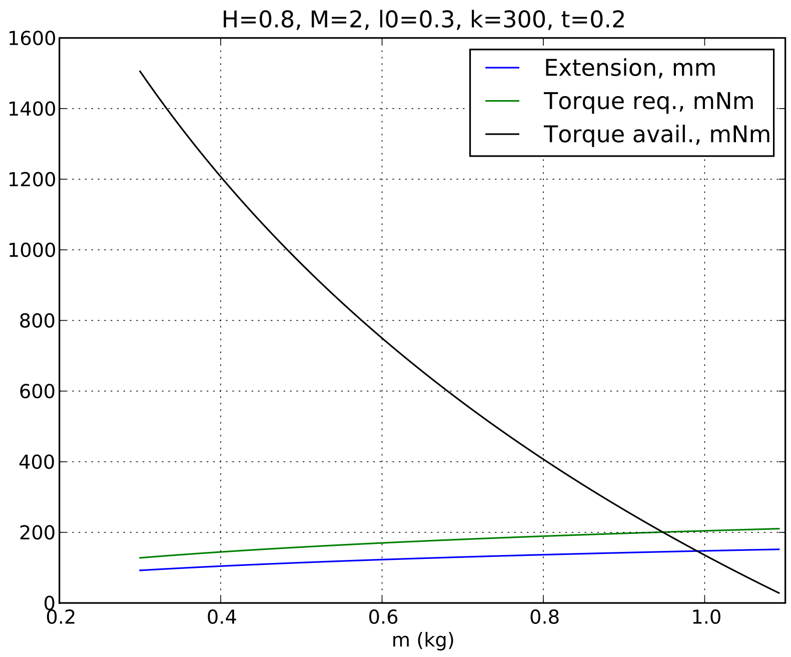


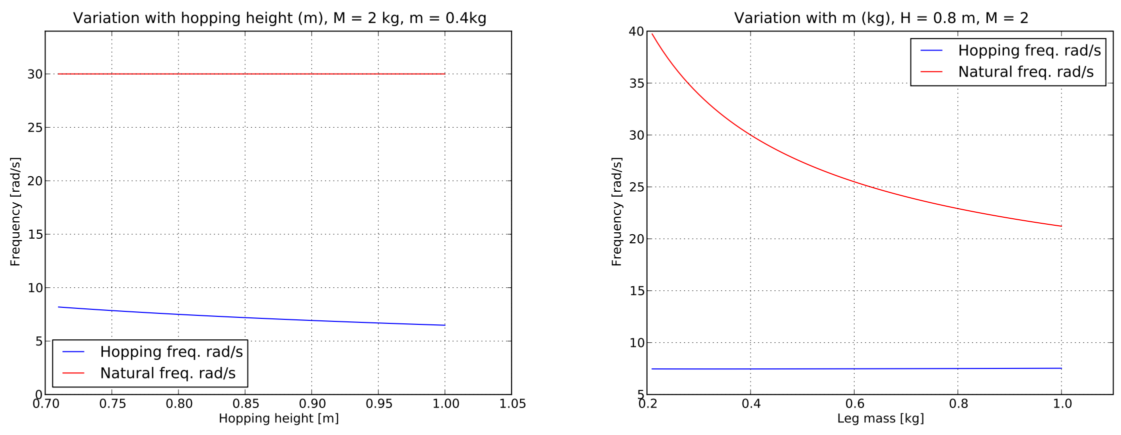
Figure 2.4: Torque variation with m

Fig. 2.4 shows the required torque for extending springs to compensate the energy lost during impact fully within 0.2 sec. It is observed that m is the single most important parameter in hopper design. The required torque is a very strong function of the leg mass. As this mass increases, we need a larger motor to satisfy torque requirements. It is noted that a value of about 0.4–0.6 kg can be called a reasonable estimate for the leg mass as we can easily choose a motor delivering the required torque for these values.

It is also seen that an extension of about 11 cms with a single spring of k = 300 Nm is needed to compensate the energy loss for hopping heights of 80 cms. So we should ensure that we can provide a maximum extension around 15 cms.

Impact Analysis

The desired hopping height dictates a hopping frequency. Intuitively, smaller hopping height results in large number of impacts per time and consequently in larger energy loss per unit time. This is seen from Fig. 2.5a because the hopping frequency is closer to the natural frequency for small hopping heights. However, beyond this consideration, since the hopper is a spring mass system, it possesses a natural frequency of its own. If the hopping frequency is near to this natural frequency, a large amount of energy is taken away by impact forces in every cycle. We intend to arrive at a range of values for the masses to ensure a large difference between the hopping frequency (ω_{hop}) and the natural frequency (ω_{nat}). Fig. 2.5b succinctly depicts all



(a) Freq. variation with height

(b) Freq. variation with m

Figure 2.5: Impact Analysis : Frequency variation

the above analysis. As the leg mass increases, the hopping frequency goes closer to the natural frequency i.e. more impact per unit time. To compound matters, more and more energy is lost per impact. So the conclusion from impact analysis is that the leg mass should be as low as possible. It is also seen from Fig. 2.5b that m = 0.4 − 0.6 kg is a good solution as well as an achievable one.

Reaction Wheel

For achieving a running gait with the hopper, it has to be started with the exact initial pitch and horizontal velocity. For any other initial condition, the hopper is pitch unstable and will not be able to continue the running gait. As mentioned in [10], an offset mass acts as a passive stabilization to the pitch attitude of the hopper. To get rid of this need for exact initial condition which is quite impractical, we design a reaction wheel on the hopper. This will result in torque coupling on the pitch axis and thus provide an active control over the pitch of the robot.

We consider the case where the pitch is such that we have no horizontal velocity and no stabilization impulse from the ground. This pitch is reoriented to 30 degrees within one hop which corresponds to a huge horizontal velocity of 13.5 m/s. The stable pitch will be less than this for lower velocities. In actual operation there will be large reaction wheel torques only while converting the initial condition into a stable running gait. After that there will only be small control torques about the stable pitch angle. Figs. 2.6 has been plotted for distance of C.G. =

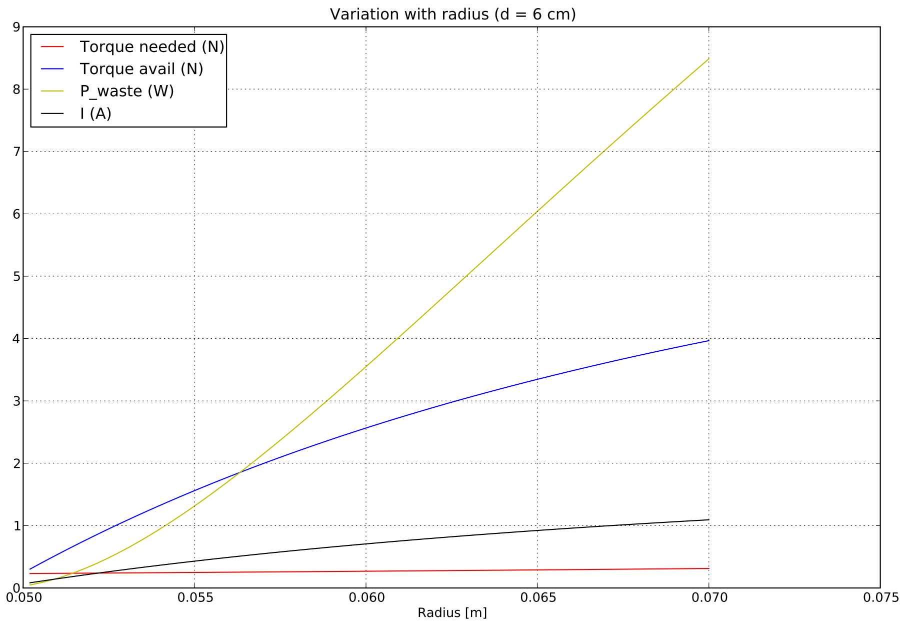


Figure 2.6: Torque requirements vs radius

6 cms and radius of the reaction wheel taken as 6 cm (mass = 1.5 kg) after a similar analysis for variation in CG. We can see that the required torque for reorientation as mentioned in above is around 500 mNm with output power being around 1.5 W.

2.3 Components

Motors

Reaction wheel motor 3257CR024 motor is chosen with a standard gear box of 14.4 : 1. The stall current is below 5A and can be easily handled with the motor drivers designed using MOSFETs with 512 cpr encoders.

Spring extension motor We take common values of worm-worm wheel diamter ratio (0.5), pressure angle (20 deg), helix angle for worm (25 deg) and co-efficient of friction $\mu = 0.3$. The torque required from the motor is not more than 200 mNm. The total power required for this task is about 4.5 W. We choose the 2342CR024 motor for this task. The gear box is taken as a 43 : 1 with 512 cpr encoders.

Servo 3003 This is used for the ratchet and paul mechanism as well as the motor sleeve disengagement. The torque requirements are satisfied easily since not much torque is needed.

Motor Driver The stall current of both the motors is near 5A and hence a conventional readymade H-bridge will not work. I designed a H-bridge using MOSFETs (CSD 16404) and MOSFET drivers (TPS 2836) to provide good drive to the motors with a maximum current of 21A and $R_{GS} = 4.1\ \Omega$. It can be controlled using just two microcontroller pins.

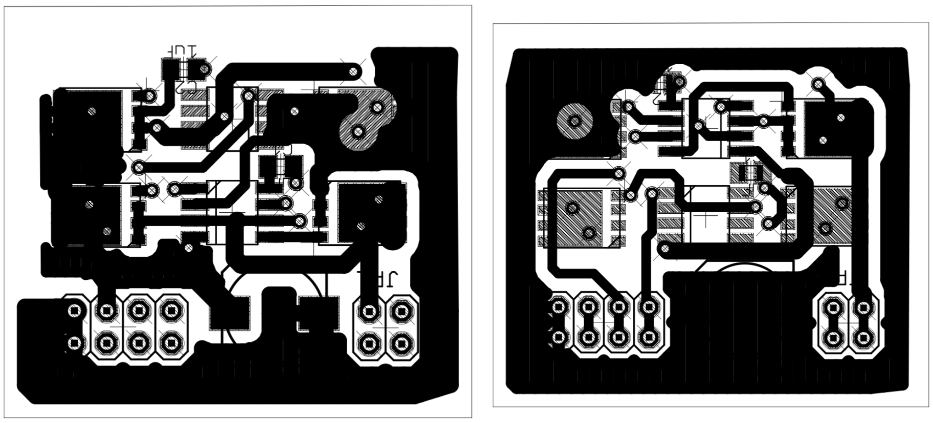

(a) Top side (b) Bottom side

Figure 2.7: MOSFET Hbridge

Inertial Measurement Unit

Accelerometer - ADIS 16201 Provides 14-bit signed readings. The accelerometer has a sensitivity of 2.162 LSB/mg with a noise of 22 LSB. The accelerometer also consists of an inclinometer to measure the angle with respect to the ground. However, it provides a sensitivity of only 10 LSB/deg. This necessitates the need of onboard inverse tangent tables to get the pitch angle from accelerometer readings. The bandwidth of the accelerometer is 2.25 KHz.

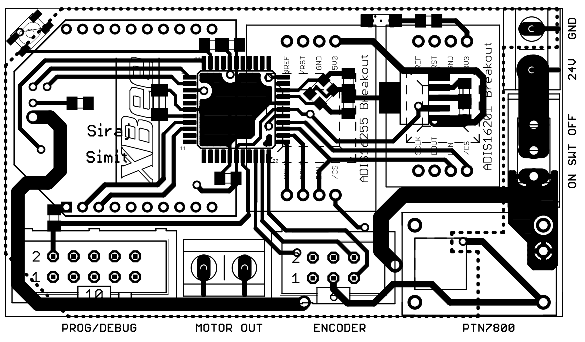
Single Axis Gyroscope - ADIS 16255 Provides 14-bit signed readings with internal temperature calibration. The sensitivity of the gyroscope is 0.07326 ^{o}/s/LSB for the whole range of $\pm$ 320 ^{o}/s with a noise of 0.48 ^{o}/s. The bandwidth of the gyroscope is 50 Hz which severely limits the update rate of the filter.

Computing

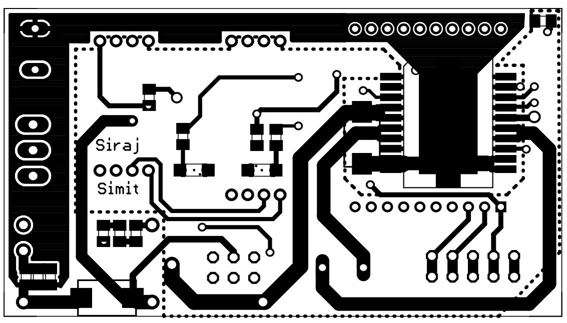
Microcontroller Microchip dsPIC33F64MC804 can run at 40 MIPS with an onboard flash memory of 64 KB along with a 16 KB SRAM. There are a host of integrated peripherals like Serial Peripheral Interface (SPI), UART, Analog to Digital converter (ADC), Quadrature

Encoder (QEI) and timers to generate Pulse Width Modulation (PWM) that can be used for motor control. Pickit 2 is the USB programmer used to program the flash of the microcontroller.

XBee Modules Used to provide a wireless link between the embedded system and the PC with the help of a custom made FT232 USB-UART converter. This is a major tool for all debugging and grabbing telemetry.



(a) Top Side



(b) Bottom Side

Figure 2.8: On-board controller pinouts

Fig. 2.8b and 2.8a show the onboard controller that was used for the experiements. A new board incorporating all drives and motors has been designed but not fabricated yet.

Chapter 3

Hopping Gait

We plan to generate a stable hopping gait by utilizing the SLOM concept for the system designed in Chapter. 2. The strategy to do that would be to first derive the equations of the motion of the whole system and propagate them using correct constraints and event solvers. The initial part of this chapter addresses these issues. Once we can propagate these set of equations from any initial condition, we try to find a relation between the impact states of every hop. The variation of these impact states will enable us to find if the gait is stable or not. We will try to find out good sets of initial conditions such that the impact states do not change much i.e. impact states are the fixed point of the Poincare Map of the set of differential equations.

Next step in the analysis would be to devise a control strategy for attitude re–orientation. The spring extension controller is already devised while finding the impact states.

3.1 Euler-Lagrange equations

We use the Euler–Lagrange equations derived from the Lagrangian as follows,

$$T = \frac{1}{2} \left[m_w \, (\dot{x_w}^2 + \dot{y_w}^2) + m_p (\dot{x_p}^2 + \dot{y_p}^2) + m_l \, (\dot{x_l}^2 + \dot{y_l}^2) + J_w \, (\dot{\phi} + \dot{\theta})^2 + J_b \, \dot{\theta}^2 \right] \tag{3.1}$$

$$V = g \, [m_l \, y_l + m_w \, y_w + m_p \, y_p] + \frac{1}{2} K \, (l - l_0)^2 \tag{3.2}$$

where,

$$\begin{aligned} x_l(t) &= x(t) + [\, d - (\, l(t) - h \,) \; \tan\theta \,] \cos\theta \\ y_l(t) &= x(t) + [\, l(t) - h] \; \cos\theta + d \; \sin\theta \end{aligned}$$

$$\begin{aligned} x_w(t) &= x(t) - \; (\, d_w - d \,) \; \cos\theta \\ y_w(t) &= y(t) - \; (\, d_w - d \,) \; \sin\theta \end{aligned}$$

$$\begin{aligned} x_p(t) &= x(t) + d \; \cos\theta \\ y_p(t) &= y(t) + d \; \sin\theta \end{aligned}$$

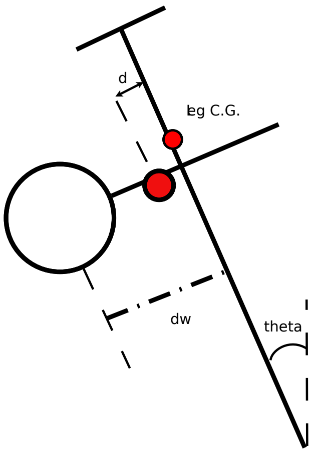


Figure 3.1: Line sketch for the hopper

x_w, y_w	:	co-ordinates of the reaction wheel
x_l, y_l	:	co-ordinates of the C.G. of the leg
x_p, y_p	:	co-ordinates of the platform
x, y	:	co-ordinates of the C.G. of the whole robot
$l(t)$	:	length of the spring
θ	:	pitch angle measured as shown in the figure
ϕ	:	angle rotated by the reaction wheel relative to the robot
d	:	distance of C.G. from the line of impact force
d_w	:	distance of reaction wheel from the line of impact force
l_0	:	non–extended length of the spring

For co-ordinates of $q = [\,x(t) \quad y(t) \quad l(t) \quad \theta(t) \quad \phi(t)\,]$, we can derive the general equations of motion of the hopper by $L = T - V$,

$$\frac{d}{dt}\left(\frac{\partial\, L}{\partial\, \dot{q_A}}\right) - \frac{\partial\, L}{\partial\, q_A} = Q_A \tag{3.3}$$

Q_A is called the generalized constraint force. If

$$\psi_i(q) = 0, \;\; i = 1, \ldots, m \tag{3.4}$$

are m constraint equations, we have,

$$Q_A = \sum_{r=1}^{m} \lambda_r \,\frac{\partial \psi_r}{\partial q_A} \tag{3.5}$$

We can divide the motion into three distinct stages,

1.	Stance Phase	: Foot in contact with ground, spring free to contract
2.	Flight Phase	: Spring is being extended, no constraints used
3.	Spring Phase	: Spring length constant

Note that all algebra is solved symbolically in Mathematica. The equations thus derived are quite messy and hence not included in the report. The script to derive them is given in the appendix.

3.2 Phases of motion

Let us assume for some time that we do not have reaction wheel control and thus $\phi(t) = 0$ is always true. This is converted into another constraint for every phase of motion and used.

Stance Phase

We define this phase from the moment the foot of the robot touches the ground till lift–off. Certain points to note are,

1. Foot touches the ground when,

$$y(t) = (\ l(t_{impact}) - l_0\ )\ \cos\theta_{impact} + d\sin\theta_{impact} \tag{3.6}$$

   This is the condition when the previous phase i.e. spring phase ends.

2. The impact torque acts upon the hopper C.G. We can find the moment arm of this torque and conserve angular momentum to get the final pitch rate.

$$arm = d\cos\theta + (\ l(t_{impact}) - l_0 - d\tan\theta\ )\ \sin\theta \tag{3.7}$$

$$\dot{\theta}_{new} = \dot{\theta}_{old} - (m_w + m_p + m_l)\ \frac{\dot{y}(t_{impact})\ arm}{J_b} \tag{3.8}$$

3. The constraint equations for this are,

$$y(t) = (\ l(t) - l_0\ )\ \cos\theta + d\sin\theta$$
$$x(t) = x_{f-impact} - (\ l(t) - l_0\ )\ \sin\theta - d\cos\theta$$

   where

$$x_{f-impact} = x(t_{impact}) + (\ l(t_{impact}) - l_0\ )\ \sin\theta_{impact} + d\cos\theta_{impact} \tag{3.9}$$

4. For deriving the equations, we follow a slightly different approach. Instead of solving for the Lagrange multipliers as described earlier, we can substitute these constraint equations in the Euler-Lagrange equations to get the final set of equations. Note that substituting them in the lagrangian is not correct as the derivation of E-L equations assumes that the variables are independent.

5. Propagate the three differential equations thus obtained to stop using an event locator. We have designed the hopper in such a way that when the spring is at its normal length, the platform touches the top portion (from which springs are suspended). We want the platform to hit the robot with the maximum velocity to ensure maximum energy transfer. Hence, the event locator should stop the equation propagation at $l(t) = l_0$.

6. We assume an inelastic collision between the platform and the robot i.e. they travel together after impact. This is ensured in the design by the ratchet and paul mechanism. Conserve momentum again to arrive at the final velocity of the robot.

Flight Phase

This stage is defined from lift–off till the spring extension is complete.

1. Extension of the spring takes place during this phase. I have used a triangular velocity profile for the energy pumping motor. This means a constant positive and negative acceleration for the first half and the second half respectively.

$$\ddot{l}(t) = \begin{cases} 0 & l(t) \leq (l_0 - \epsilon) \;\; OR \;\; l(t) \geq (l_{max} + \epsilon) \\ \\ l_{accel} & l_0 \leq l(t) \leq \frac{(l_{max}+l_0)}{2} \\ \\ -l_{accel} & \frac{(l_{max}+l_0)}{2} \leq l(t) \leq l_{max} \end{cases} \tag{3.10}$$

A buffer of ϵ is necessary because Mathematica uses floats and sometimes, the software calculates the value of $l(t)$ very close to l_{max} and returns the comparison $l(t) == l_{max}$ as true even though there is a very small difference between them.

2. We have used $\ddot{l}(t)$ in the control law. This means that the torque of the motor is being actuated. We need to convert this into a form that is implementable using the onboard rotatory encoders. This is done as follows,

   - Start onboard time at lift-off. Calculate the desired velocity of the EPM motor according to Equation 3.10. This is the desired $\omega_d(t)$ of the motor.
   - If $\omega(t)$ be the speed sensed by the encoders, we can devise a PID controller as,

$$e(t) = \omega(t) - \omega_d(t) \tag{3.11}$$

$$U_l(t) = K_w \, \omega_d(t) + K_p \, e(t) + K_d \, \frac{d\, e(t)}{dt} + K_i \int e(t) \, dt \tag{3.12}$$

   K_w is the speed constant of the motor.

3. Flight equations are completely unconstrained.

4. The event solver stops the propagation of flight equations when $l(t) = l_{max}$.

Spring Phase

This is the remainder of the hopper motion in air till the foot touches the ground.

1. Spring length is constant during this phase. The constraint equation is therefore,

$$l(t) = l_{max} \tag{3.13}$$

2. The event solver for spring phase stops integration when,

$$y(t) = (\ l(t) - l_0\ )\ \cos\theta + d\sin\theta \tag{3.14}$$

Figure. 3.2 shows these equations being used to together to generate a hopping gait. We have started the hopper from a good initial condition arrived at by methods discussed in later sections and the figure shows the trajectory of the C.G. during the motion. Figure 3.3 shows the variation of states during the Spring Phase. It is seen that attitude re–orients itself due to SLOM effect during this.

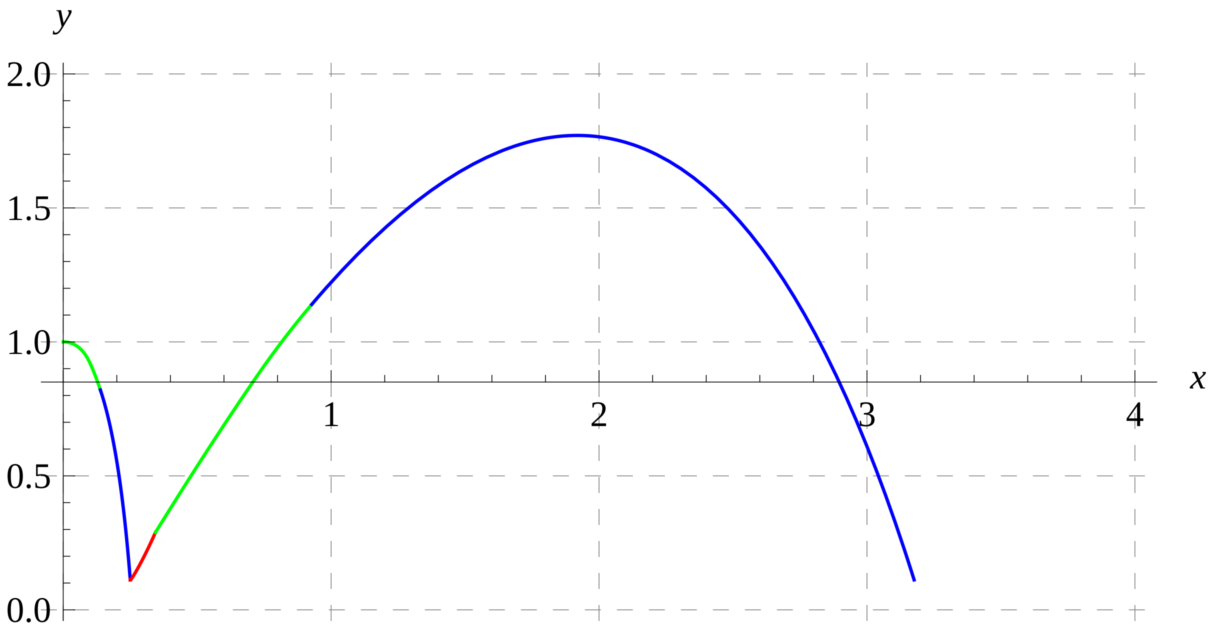


Figure 3.2: Propagation of equations of motion : Flight Phase - Green, Spring Phase - Blue, Stance Phase - Red, distances in meters

3.3 In-place hopping

It is difficult for SLOM system to perform inplace hopping. The impact torque is a destabilizing one and hence active control is necessary for inplace hopping. Figures 3.6, 3.4 and 3.5 show the performance of the controller described below for inplace hopping.

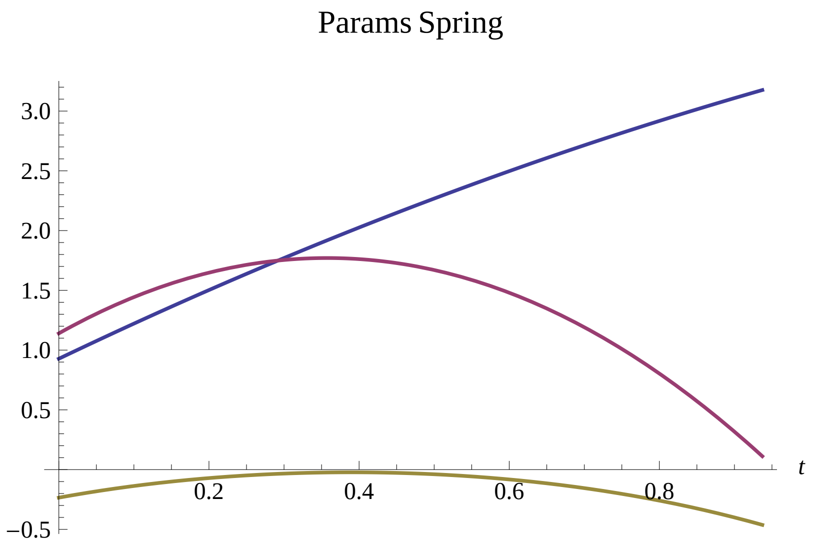


Figure 3.3: Spring Phase parameters : $x(t)$ - Blue, $y(t)$ - Pink, $\theta(t)$ - Yellow, t = secs

1. The basic characteristic of the controller will have to be that it should always keep the attitude of the robot constant. The value of this constant attitude needs to be found out by a detailed analysis. We would like to take into account the energy spent by the controller to keep the attitude constant inorder to decide the value of the desired attitude. The cost metric can be the deviation of the horizontal distance in successsive hops and the energy consumed ($\dot{\theta}\ \ddot{\theta}$). I decided to keep the attitude constant to zero for the time being.

2. The flight and spring phases consist of a simple PID controller that re-orients the attitude to bring it to zero. Figure 3.4 shows its performance after the hopper is released with an attitude of $1rad$. The controller manages to quickly correct the attitude.

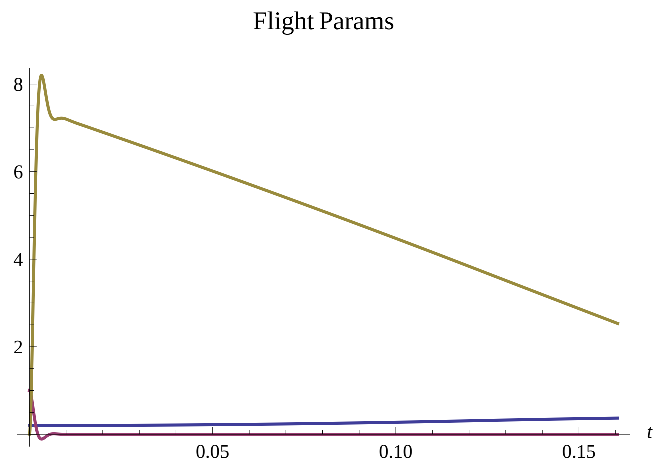


Figure 3.4: Inplace Flight Phase parameters : $l(t)$ - Blue, $\theta(t)$ - Pink, $\phi(t)$ - Yellow, t = secs

3. The stance phase controller is slightly different. It was observed that if the desired attitude

was zero in the stance phase also, the hopper slowly starts to drift backwards in successive hops. This is rectified by giving the desired attitude after the stance phase as,

$$\theta_d = \tan^{-1}\left(\frac{x_{impact}}{h_{max}}\right) \tag{3.15}$$

4. This ensures that the hopper always tries to go towards $x = 0$ position and thus performs satisfactory inplace hops. The performance of the stance controller is shown in Figure 3.5

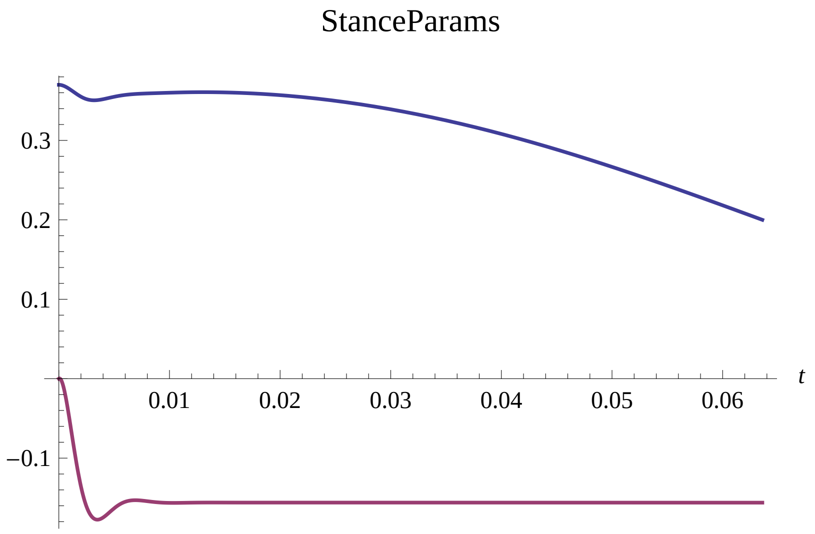


Figure 3.5: Inplace Stance Phase parameters : $l(t)$ - Blue, $\theta(t)$ - Pink, t = secs

3.4 Stable gait

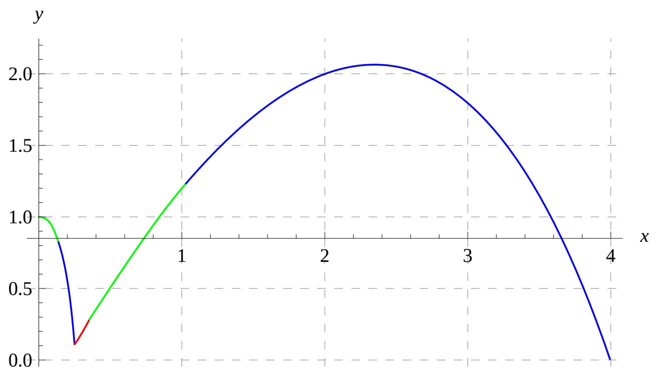


Figure 3.7: Propagation of equations of motion with $\Delta\ \dot{\theta}(t) = 0.05\ rad/s$

Figure. 3.7 shows the change in trajectory after a small perturbation in the initial conditions at $t = 0$ i.e. at $x = 0$ in $\dot{\theta}$. Note that the maximum height is significantly higher and the height of the second impact point is almost zero, this suggests a very bad pitch attitude of the robot

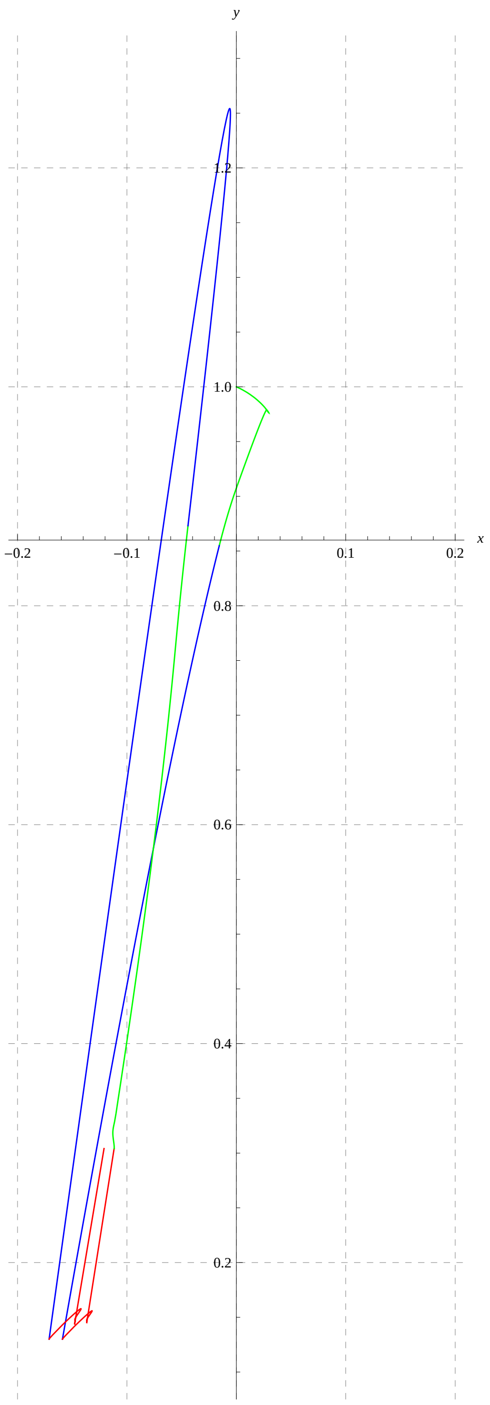


Figure 3.6: Trajectories of two inplace hops : Flight Phase - Green, Spring Phase - Blue, Stance Phase - Red, distances in meters

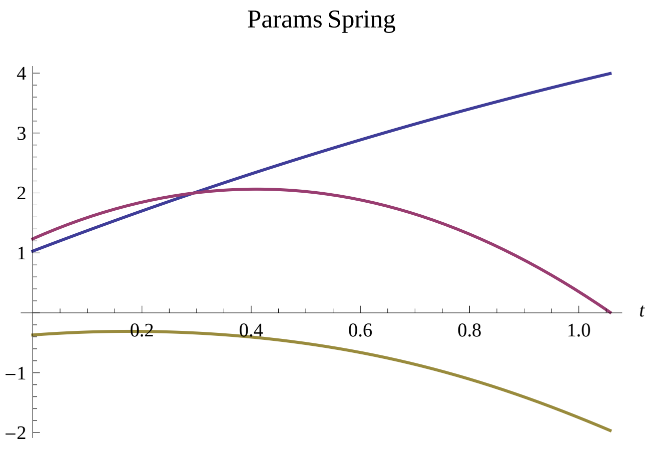


Figure 3.8: Spring Phase parameters : $x(t)$ - Blue, $y(t)$ - Pink, $\theta(t)$ - Yellow, t = secs

as is verified by Figure. 3.8. The hopper falls flat on its face! Thus, our next aim is to find out a set of initial conditions such that the states at the point of impact at successive hops do not differ by much. We define the measure of the change in the impact states by taking the norm of the states q_A and q_B,

$$norm = \sum_{i=1, i \neq 1,5,10}^{10} ||q_{A_i} - q_{B_i}|| \tag{3.16}$$

where $q = [\, x \;\; y \;\; l \;\; \theta \;\; \phi \;\; \dot{x} \;\; \dot{y} \;\; \dot{l} \;\; \dot{\theta} \;\; \dot{\phi} \,]$ and we remove $x(t)$, $\phi(t)$ and $\dot{\phi}(t)$ from the norm.

Good initial conditions

The only thing we can control is the initial conditions at the launch. Everything after that is a product of the propagation of differential equations. We try to find a stable gait as follows,

1. Keep desired parameters as a hopping height of 1 m and horizontal velocity of 1 m/s. Start propagation of the equations from the topmost point of the trajectory, i.e. $x = 0$, $\dot{x} = 1$, $\dot{y} = 0$ and $y = 1$. ϕ and $\dot{\phi}$ are of course both zero.

2. Now we have l_{max}, θ_0 and $\dot{\theta}_0$ as the three parameters which we can vary.

3. Since we are neglecting friction in all the above analysis, we can calculate the amount of energy lost in the impact and thus the energy needed to be put into the system each hop. This gives, $l_{max} = 0.37m$.

4. We now try to run an optimizing algorithm to get optimal states at the launching instant as shown in [22]. Let the attitude of the robot be zero at the launch. This fixes θ_0. The only parameter left to be varied is $\dot{\theta}_0$.

5. We are using Mathematica to compute all this. There are a host of problems associated with it. It is slow to propagate the equations and they also become stiff at certain initial conditions when the solver fails.

6. An optimizer will try to find a global minima for the objective function (using the internal NMinimize method) which is the minimization of norm in our case. It needs to compute these equations for a lot of different initial conditions quickly which Mathematica cannot do. Different approaches like Genetic Algorithms other than the default built-in optimization routine are also equally slow.

7. On top of this, the problem with our system is that it is a very rapidly changing system. There are a lot of local minimas very near to each other. NMinimize gets stuck in these minimas and it jumps out of one to get stuck in the neighbouring one.

8. So instead of using NMinimize to find states such that the norm goes to zero, I tried to iterate over the region manually. By oberserving the norm at each stage, I found out a few sets where the norm was small enough to be taken care of by the controller. The trajectory generated by these conditions is showed in Figure 3.2.

3.5 Poincare Map

It is known that our system is a highly non-linear periodical dynamical system. Poincare Map of *first return map* is one classical tool to study the dynamics of periodic systems. Consider a set of autonomous differential equations,

$$\frac{d\mathbf{x}}{dt} = \dot{\mathbf{x}} = f(\mathbf{x}) \tag{3.17}$$

where $\mathbf{x} = \mathbf{x}(t) = [x_1, \ \ldots \ , x_n]$ is the state vector of independent variables. $f : \mathcal{U} \to \mathcal{R}^n$ where $\mathcal{U} \subseteq \mathcal{R}^n$. Let $\phi_t(\mathbf{x}) = \phi(\mathbf{x}, t)$ be a flow generated by vector field f and satisfies Equation 3.17. An initial condition is $\mathbf{x}_{in} = \mathbf{x}(0)$ and the corresponding solution is $\phi(\mathbf{x}_{in}, t)$ with $\phi(\mathbf{x}_{in}, 0) = \mathbf{x}_{in}$. We now have two curves, a solution curve that propagates the solution given an initial condition $\mathbf{x}_{in}$ and an orbit curve which is a set of all the states taken by this solution curve. The kind of solutions we are interested in are,

$$f(\bar{\mathbf{x}}) = 0 \tag{3.18}$$

This is the fixed point of the solution which might have the following characteristics,

1. The solution might start in the neighbourhood of $\bar{\mathbf{x}}$ and continue to remain near it. This is called a *stable* solution.

2. The solution might converge asymtotically towards $\bar{\mathbf{x}}$ to give an *aymsptotically stable* solution.

3. It can diverge from $\bar{\mathbf{x}}$ if $\bar{\mathbf{x}}$ is an unstable solution.

We can thus see that a Poincare Map converts a n^{th} order dynamical system into a $(n-1)^{th}$ order discrete system. It is almost perfect for our analysis. We can examine the periodicity and the stability of the locomotion with the help of this map.

Let γ be a closed orbit of the n dimensional system. Find a local cross-section about γ of dimension $(n-1)$ and let it be Σ. Σ is the (n-1) dimensional hyper-surface. It need not be planar but has to be transverse to γ. Consider a point $\mathbf{p}$ on Σ. All orbits near γ have to pass through Σ. If we define Σ such that it is a smooth scalar function $g : \mathcal{R}^n \to \mathcal{R}$, we can say,

$$\Sigma = \{\mathbf{x} \in \mathcal{R}^n : g(\mathbf{x}) = 0\} \tag{3.19}$$

The condition on g is just that

$$\nabla g(\mathbf{p})^T f(\mathbf{p}) \neq 0 \tag{3.20}$$

i.e. the gradient is not orthogonal to the flow at point p. Let $\mathcal{U}$ be a neighbourhood of p such

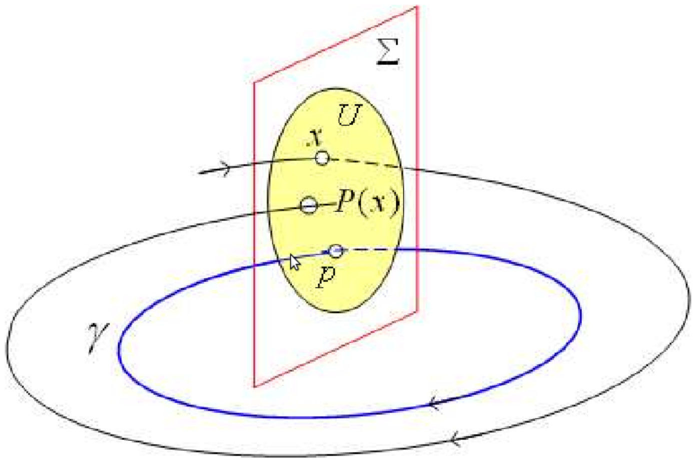


Figure 3.9: Poincare Map : Schematic [11]

that γ intersects Σ only once in the neighbourhood. We define the Poincare map from $\mathcal{U} \to \mathcal{R}^n$ as,

$$\mathbf{P}(\mathbf{x}) = \phi_\tau(\mathbf{x}) \tag{3.21}$$

ϕ_τ is the flow of the system. As the point $\mathbf{x} \to \mathbf{p}$, $\tau \to T$ which is the return period of the system at $\mathbf{p}$. Thus, we can say,

$$\mathbf{x}_{k+1} = \mathbf{P}(\mathbf{x}_k) \tag{3.22}$$

If the Jacobian of this map,

$$\mathbf{J_p} = \frac{d\ \mathbf{P}(\mathbf{x})}{d\mathbf{x}} \quad at \ \ \mathbf{x} = \mathbf{p} \tag{3.23}$$

has all its eigenvalues inside the unit circle, the point $\mathbf{p}$ is a stable point.

For our system, we try to find a map between successive impact states. The fixed point of the map will be an equilibrium point and can be analysed for stability. The Poincare Map is thus a routine which takes in $\mathbf{x}_k$ and gives out $\mathbf{x}_{k+1}$ after propagating the differential equations.

The optimization procedure described above also aims to find the fixed point. So using exactly similar procedure, we arrive at a set of states such that the robot hops for around 2 hops without falling. Note that this is not the exact fixed point, it has been been arrived at manually. We perturb the initial states by a small amount to see that the final impact states change drastically. Hence, this is in the neighbourhood of an unstable fixed point. Due to issues mentioned above, I have not been able to find a stable fixed point for the hopper. All the points I found have non-zero values of norms although they are very small.

```
poincareMap[{t0_, x0_, y0_, len0_, theta0_, xdot0_, ydot0_,
lendot0_, thetadot0_}] := Module[{pInit},

pInit = {t0, x0, y0, len0, theta0, xdot0, ydot0, lendot0, thetadot0};
{op1, pt1, pt2} = stancePhase[pInit];
{op2, pt1, pt2} = flightPhase[op1];
{op3, pt1, pt2} = springPhase[op2];
op3[[1]] = 0;
op3[[2]] = 0;

op3
];

cost[op1_List, op2_List] := Module[{op},
op = 0;
For[i=3, i <= 9, i++, If[i == 5, Indeterminate, op = op + (op2[[i]] - op1[[i]])^2];
Sqrt[op]
];
```

3.6 Control Strategies

We have the reaction wheel and the spring length as two control variables. The control of spring length is done as described in Section 3.2. We look at orientation control in this section. We would like to achieve trajectory following using orientation control.

Stance Phase

A few points to be noted are,

1. Stance Phase is the most important phase for trajectory following. A bad attitude at lift-off results in a totally different trajectory in space and mere orientation control is obviously not enough to follow it.

2. We decide to follow the time series of attitude values during stance phase. Stance times are small, about 30-40 msecs. The control procedure is as follows,

    - Generate a time series of the attitude using the good launch parameters and call it $\theta_G(t)$. Perturb the launch parameters by a small amount and start the integration to get a time series $\theta(t)$.

- Define,

$$e(t) = \theta(t) - \theta_G(t) \tag{3.24}$$

  This gives $\frac{de}{dt} = \dot{\theta}(t) - \dot{\theta}_G(t)$.
  We also put in the integral gain as $iErr = \sum e$. It is ensured that,

$$-iErrMax \quad \leq \quad iErr \quad \leq \quad iErrMax$$

- The control input is given as,

$$\ddot{\phi}(t) = K_p\, e + K_d\, \dot{e} + K_i\, iErr \tag{3.25}$$

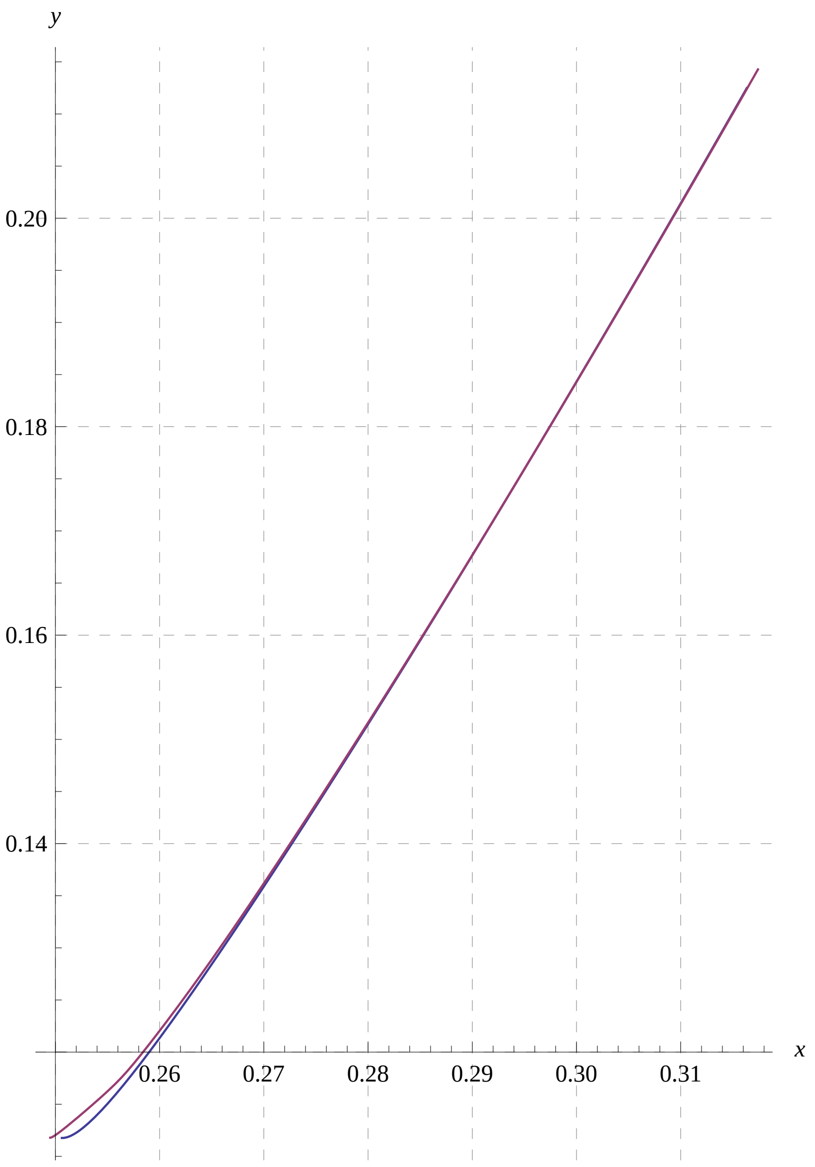


Figure 3.10: Stance Controller : Good trajectory - Blue, Perturbed trajectory - Pink, distances in meters

3. A few problems in the control strategy are apparent. The stance times of the good trajectory and the control trajectory will not be equal. Hence, a PID controller based upon the comparison on the time series will not always work.

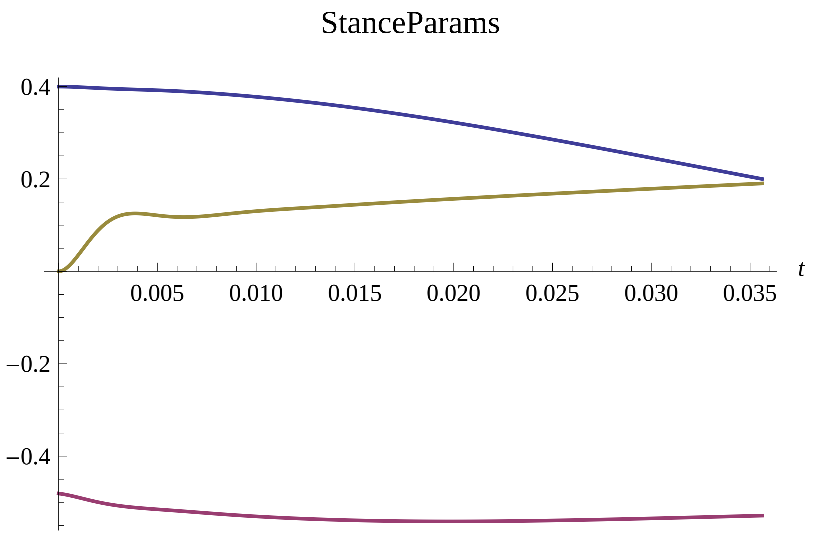


Figure 3.11: Stance Controller variables : $l(t)$ - Blue, $\theta(t)$ - Pink, $\phi(t)$ - Yellow, t = secs

Flight and Spring Phases

1. It was noted that the perturbed trajectories in the flight and spring phases are significantly different than the good ones. This is due to the large time scales associated with them, around 400–500 msecs.

2. We need a different control strategy for two reasons, firstly a slight error at liftoff causes a different trajectory in space and trajectory following breaks down, secondly due to the reason mentioned above.

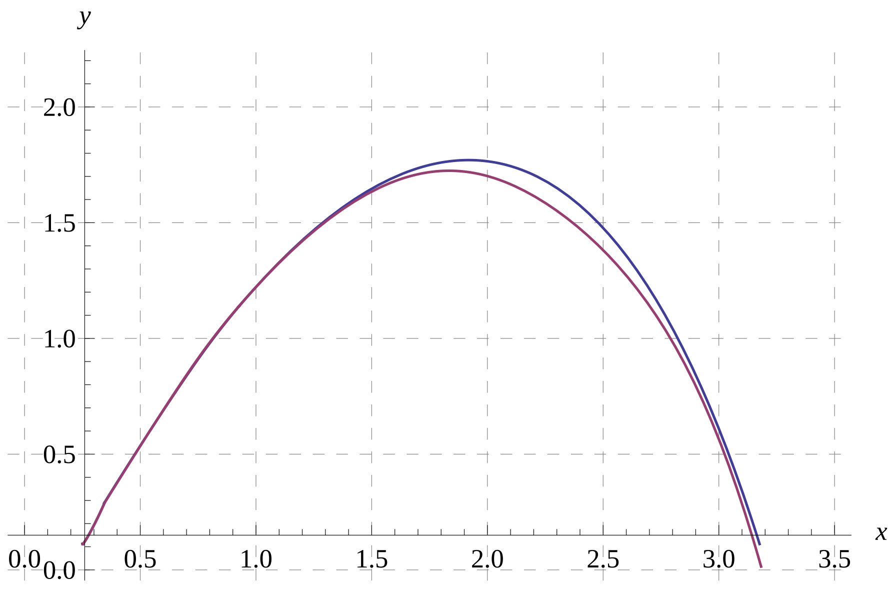


Figure 3.12: Trajectory Controller : Good - Blue, Perturbed - Pink, Note that the trajectories deviate significantly after the flight phase resulting in a relatively bad attitude at impact.

3. I tried out a different control law viz.

   - Solve the attitude re–orientation problem at every instant when the control input gets calculated to ensure that the hopper lands on the ground with the same attitude as that of the previous hop.

$$y_{impact} = (l(t_{impact}) - l_0)\ \cos\theta_{impact} - d\sin\theta_{impact} \tag{3.26}$$

   - Find the time left for impact t_{left} as a solution of,

$$y_{impact} - y(t) = \dot{y}(t)\ t - \frac{1}{2}\ g\ t^2 \tag{3.27}$$

   - Find the displacement in θ to impact with the correct attitude and give the corresponding torque command to the controller,

$$\Delta\ \theta(t) = \theta_{impact} - \theta(t) \tag{3.28}$$

$$\ddot{\phi}_d(t) = \left(\frac{-2\ J_b}{J_w}\right)\left(\frac{\Delta\ \theta - \dot{\theta}(t)\ t_{left}}{t^2_{left}}\right) \tag{3.29}$$

   - This is the reference signal for the controller. We put a PID controller on top of this where the error is taken as,

$$e(t) = \ddot{\phi}(t) - \ddot{\phi}_d(t) \tag{3.30}$$

$$U_\phi(t) = \ddot{\phi}_d(t) + K_p\ e(t) + K_d\ \dddot{\phi}(t) + K_i\ \ iErrPhi \tag{3.31}$$

     as done in the spring extension controller. This is again a torque controller and needs to be converted into a velocity controller as done in Equation 3.12.

4. As shown in Figure 3.12, the controller manages to follow the trajectory pretty well. However, the attitude at the end of the perturbed trajectory is quite bad. Further refinement in control strategy is necessary to improve performance.

Chapter 4

Attitude Estimation

4.1 Introduction

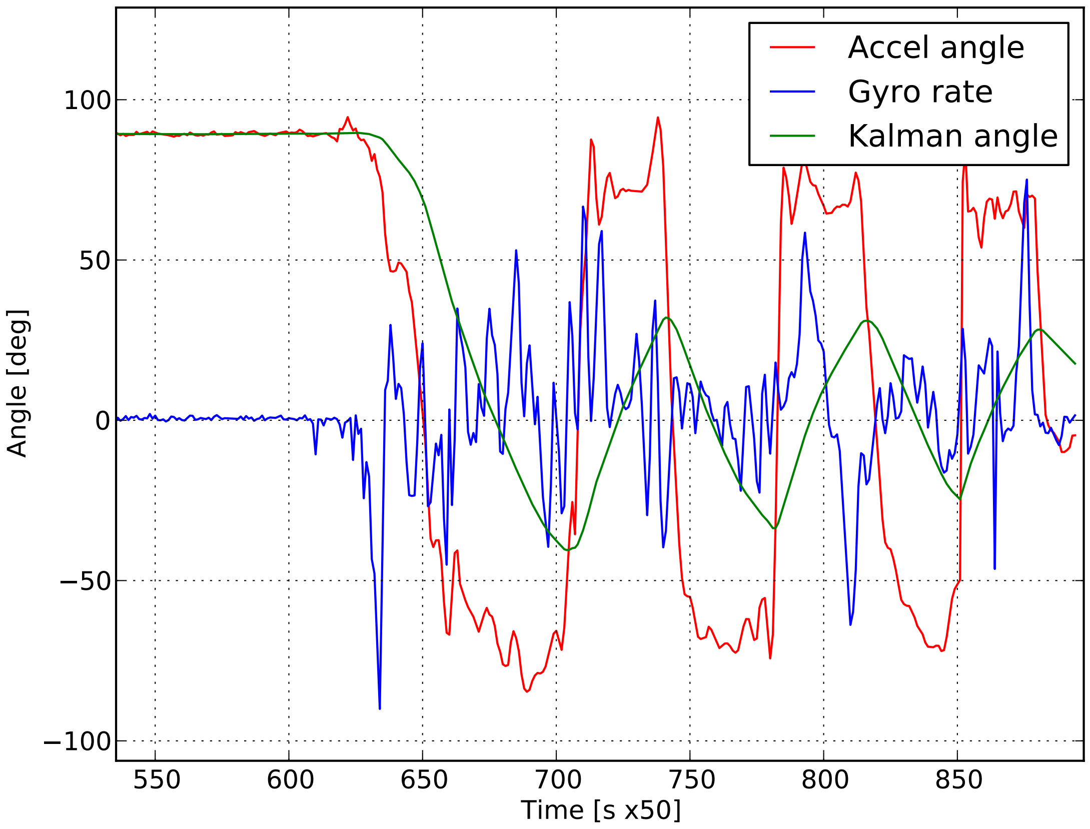


Figure 4.1: Kalman filter : Low frequency input

As shown in Fig. 4.1, the readings of the gyroscope and the accelerometer are highly noisy and erratic even in the situation of smooth movements of the IMU. This shows, that depending upon either of the two sensors is not good for pitch estimation. Gyroscopes are high frequency sensors and accurately estimate the rate. However, they also show pronounced variation of readings with time (gyro bias) and temperature (compensated to an extent). Accelerometers are good at low frequency measurements. These do not suffer from a growing bias like the gyroscopes and can be used to correct the readings estimated on the basis of gyroscope rate from time to time. We look at Kalman filter as a way to fuse these two sensors.

Various alternatives for filtering schemes exist and complementary filters are very widely used for fusing accelerometers and gyroscopes on IMUs. I decided to use a Kalman filter because the micro-controller can easily handle the computations at usable update rates of about 20-50 Hz. This is one of the major reasons cited in literature for the use of complementary filters over Kalman filters. It is also noted that for linear systems, there is little difference in the equations of the two filters.

4.2 Equations

The equations for Kalman filter with a state vector $\boldsymbol{x} = [x_1 \;\; x_2]^T = [\theta \;\; \dot{\theta}]^T$ are give below.

State equation :

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\,\boldsymbol{x}_k + \boldsymbol{B}\,\boldsymbol{u}_k + \boldsymbol{w}_k \tag{4.1}$$

Output equation :

$$y_{k+1} = \boldsymbol{C}\,\boldsymbol{x}_{k+1} + z_{k+1} \tag{4.2}$$

Update equations :

$$\boldsymbol{K}_k = \boldsymbol{A}\,\boldsymbol{P}_k\,\boldsymbol{C}^T\,\left(\boldsymbol{C}\,\boldsymbol{P}_k\,\boldsymbol{C}^T + S_z\right)^{-1} \tag{4.3}$$

$$\hat{\boldsymbol{x}}_{k+1} = (\boldsymbol{A}\,\hat{\boldsymbol{x}}_k + \boldsymbol{B}\,\boldsymbol{u}_k) + \boldsymbol{K}_k\,(y_{k+1} - \boldsymbol{C}\,\hat{\boldsymbol{x}}_k) \tag{4.4}$$

$$\boldsymbol{P}_{k+1} = \boldsymbol{A}\,\boldsymbol{P}_k\,\boldsymbol{A}^T + \boldsymbol{S}_w - \boldsymbol{A}\,\boldsymbol{P}_k\,\boldsymbol{C}^T\,S_z^{-1}\,\boldsymbol{C}\,\boldsymbol{P}_k\,\boldsymbol{A}^T \tag{4.5}$$

For our state vector, the equations consist of,

$$\boldsymbol{A} = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix} \quad \boldsymbol{B} = \begin{bmatrix} dt \\ 0 \end{bmatrix} \quad \boldsymbol{u} = \begin{bmatrix} \dot{\theta}_{gyro} \\ 0 \end{bmatrix} \quad y = \theta_{accel} \quad \boldsymbol{C} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{4.6}$$

$\boldsymbol{P}$ is called the estimation error co-variance and can be initialized to some value, a small value implies that we expect the error co-variance to be small too. We assume that the estimation errors are completely dependent upon one another and hence initialize the matrix as identity. S_z is the accelerometer variance obtained from the datasheet. $\boldsymbol{S}_w$ is the gyroscope covariance matrix. Let $\nu_{angle} = dt\,\sigma_{rate}$ and $\nu_{rate} = 0$. These values are thus obtained from the datasheet for a particular filter update rate.

$$\boldsymbol{P} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \boldsymbol{S}_w = \begin{bmatrix} \nu_{angle}^2 & \nu_{angle}\,\nu_{rate} \\ \nu_{angle}\,\nu_{rate} & \nu_{rate}^2 \end{bmatrix} = \begin{bmatrix} 92 \times 10^{-6} & 0 \\ 0 & 0 \end{bmatrix} \tag{4.7}$$

Eqns. 4.1 – 4.4 were converted to their algebraic form instead of the matrix operations for better calculation speed. Fig. 4.1 shows the performance of this filter with calculations being done on the computer. Fixed-point arithmetic has been implemented on the micro-controller and will be used for the onboard Kalman filter.

4.3 Results

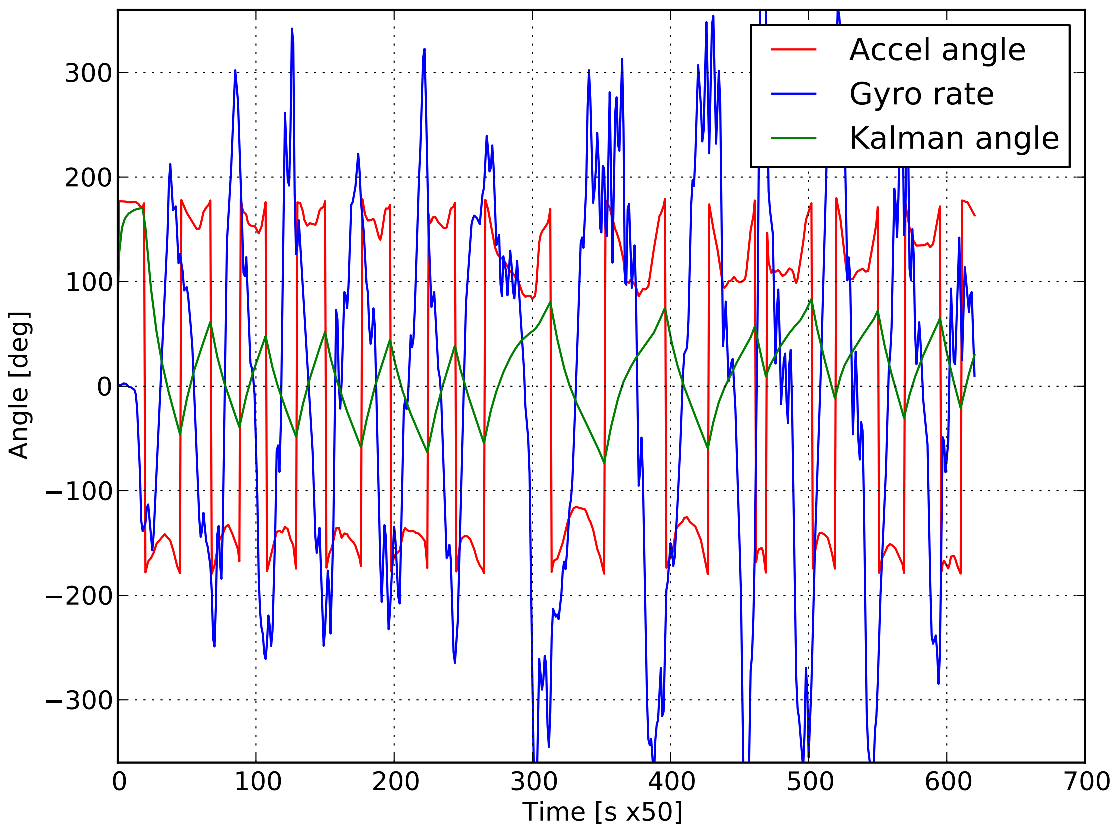


Figure 4.2: Kalman filter : High frequency input

Fig. 4.2 has been plotted for a very high frequency movement of the IMU. As shown, the rates exceed $\pm 320^o/s$ which is the maximum rate detected by the gyro. The final output of the kalman filter does match the hand movement of about 90 degrees. However, at about the 300^{th} update, it completely misses a very fast 360 deg. rotation of the IMU. This is reasonable because even the accelerometer and gyroscope do not seem to significantly register this movement.

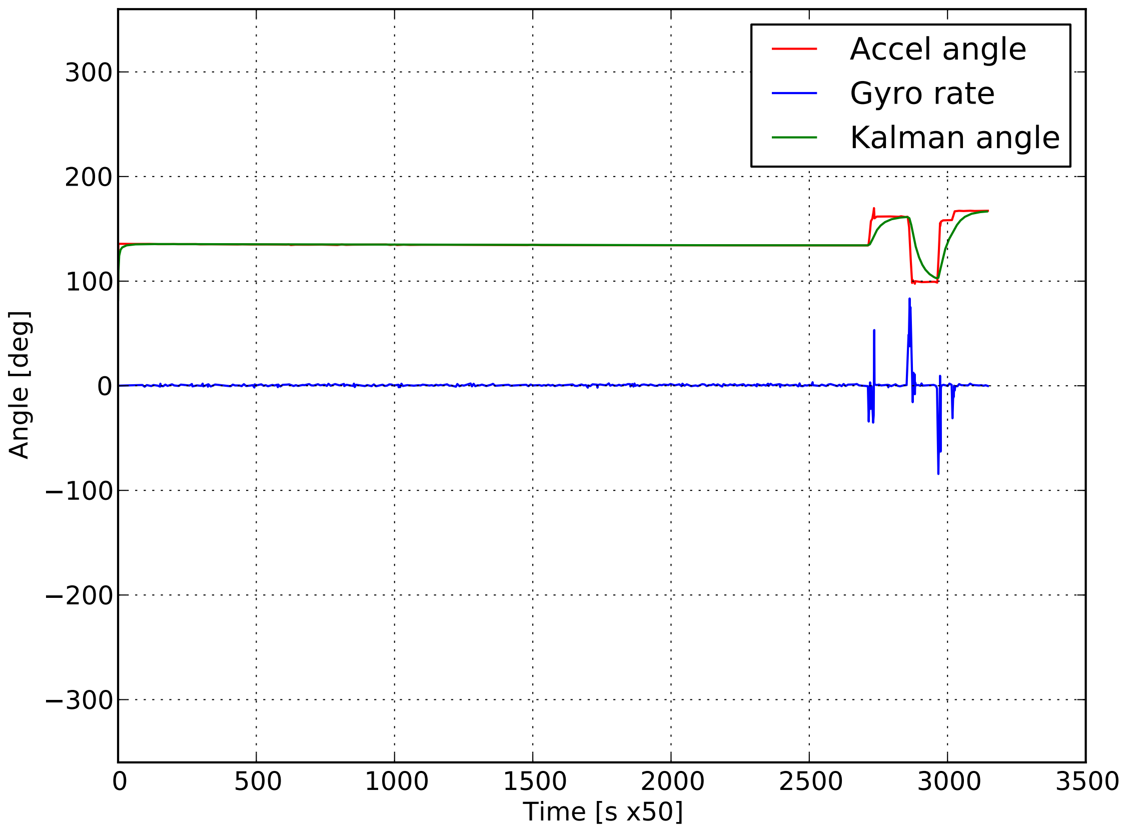


Figure 4.3: Kalman filter : Gyro drift

Fig. 4.3 shows readings plotted for 5 minutes at a 10 Hz update rate for the filter. It is seen that the gyro rate has some noise. However, integrating this noise does not show a large error in the angle moved by the sensor. The integral of the gyro rate is zero for all the time. The drift mentioned for gyroscopes is not to be seen even for periods as large as 12 minutes.

4.4 Implementation

We have to perform filtering on the onboard embedded system. It is essential that we finish off the filtering part quickly so that the microcontroller can devote enough time to implementing the control law. Polling data from the sensors also takes precious time. I set an update rate of 50 Hz as the goal, this is also the maximum update rate of the gyroscope, so the filter cannot be run faster than that. Two major parts of the implementation are as follows,

Inverse tan function

The tilt of the accelerometer is obtained as follows,

$$\theta = tan^{-1}(\frac{a_y}{a_x}) \tag{4.8}$$

The C30 library provides an implementation of the tan^{-1} function in floating point. However, these operations take time on a 16-bit microcontroller. I thus implemented it using a table of stored values and interpolating for values between them. The salient features of this are,

1. *tan* is an odd function, so we just need to include positive values in the table
2. It is highly non-linear after 70 deg. So we will have a tan^{-1} function only from -70 to +70 deg. This gives a range of 140 deg. for the attitude which will be sufficient.
3. Tangent function is linear enough till 15^o, so we use this in our calculations. Divide the interval [tan(15^o), tan(70^o)] in 16 intervals. Store these *tangents* and the corresponding *angles* in a table.
4. Given a value, find out the interval within which it lies. Linearly interpolate *tangent* function within this interval.
5. We identify a systematic"ish" error of 0.06 deg in our implementation and hence add 0.06 deg. while calculating the angle in the tabular implementation.
6. This implementation requires just 48 bytes of memory and is computationally cheap as well.

Fig. 4.4 shows the error between the tan^{-1} function of the python math library and tan^{-1} function using the table above. We can implement this on the microcontroller using fixed point arithmetic as shown in Sec. 4.4 to further reduce the computational time.

Fixed point arithmetic

There is one another way to reduce the computational overhead of the Kalman Filter. The accelerometer provides the inclination values directly. It is known that the error in these values is ±0.5 deg. Implementing the tabular tangent function and the Kalman Filter in fixed point both is an option. I however feel that the errors in the fixed point arithmetic will dominate over the inclinometer error and hence it is not a bad idea to use the inclinometer directly. This will save some CPU cycles too. Hence I implemented the Kalman Filter in fixed point arithmetic. The salient features are as follows,

1. On a 16-bit microcontroller, we use 10.6 type signed fixed point numbers. The higher 10 bits are used for the integer part of the number whereas the lower 6 bits are used for the mantissa. The resolution of this scheme is thus $\frac{1}{2^6} = 0.015625$.

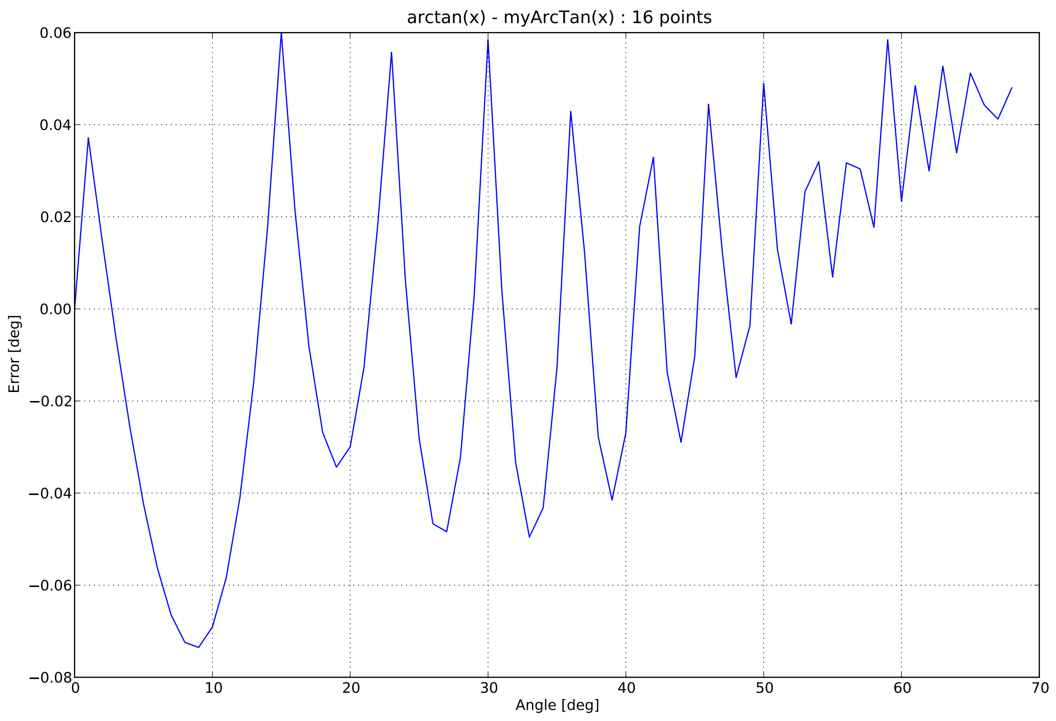


Figure 4.4: tan^{-1} function : Error between actual and the tabular implementation

2. Thus we have to scale every number by $2^6 = 64$ before operating upon it. The Kalman Filter constants are already hardcoded in the 10.6 form. The variables also operate and get operated in this form itself.

3. Multiply by 64 everytime two 10.6 fixed point numbers divide and divide by 64 everytime they get multiplied to preserve the scaling.

4. The integer part should be large enough to accomodate the rollover from the mantissa part after the multiplication.

5. Type-casting should be done properly while performing the arithmetic to prevent truncation by the compiler.

Figures 4.5 and 4.6 plot the tilt given by the accelerometers directly vs the fixed point implementation on the microcontroller vs the floating point implementation on a computer. The floating point implementation on the computer lags behind the microcontroller implementation slightly. This is probably because the covariance of the accelerometer is a floating point number with a mantissa greater than 0.5 and is thus badly rounded off in the fixed point implementation.

The source code for the fixed point implementation is given in the appendix.

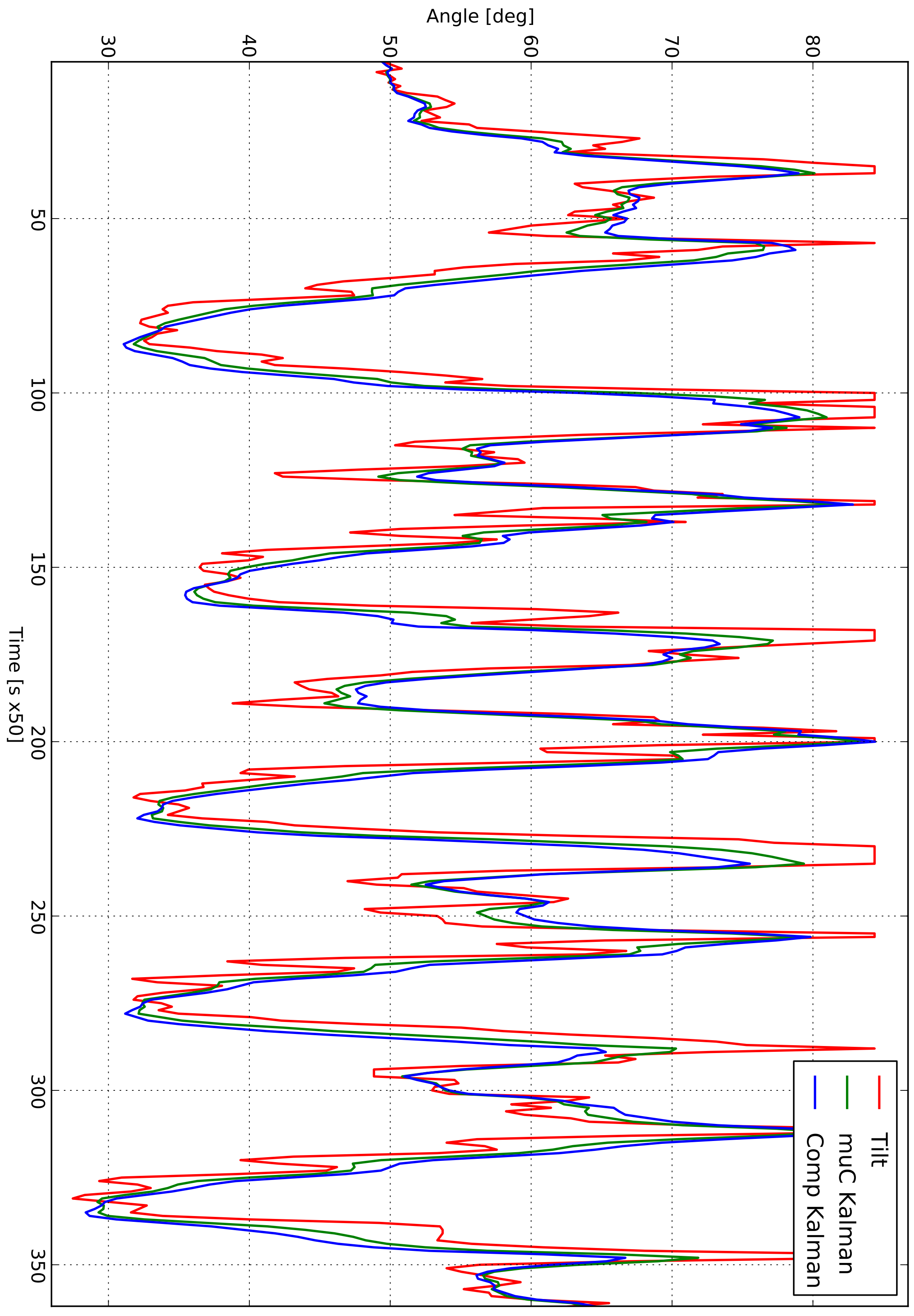


Figure 4.5: Fixed point Kalman Filter : High freqency

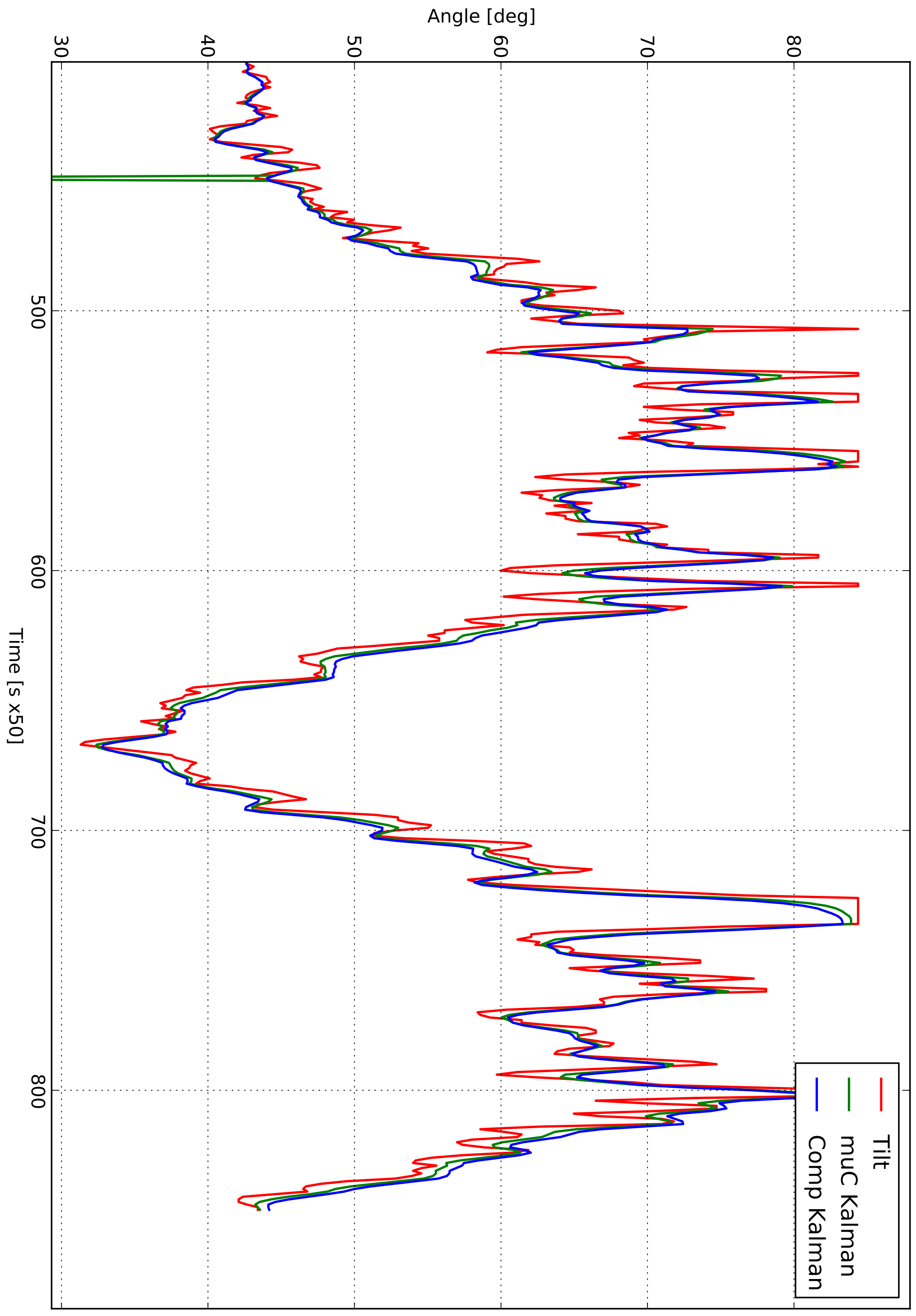


Figure 4.6: Fixed point Kalman Filter : Low freqency

Chapter 5

Conclusion

The project aimed to tackle four tasks viz.

1. The essence of this phase was to use a number of simple back-of-the-envelope style analyses to get an idea of the relative importance of the enormous number of parameters associated with this system. Such analyses were instrumental in getting an intuitive grasp of the complex dynamics without getting into mathematical complexities. It also helped a lot while performing design iterations to decide final dimensions.

2. - This concentrated on the control aspect of the robot. I have modeled the whole non-linear system for the hopping robot and simulated stable in-place hopping as well as a stable hopping gait.
   - I faced some limitations due to the nature of the computing environment (Mathematica) and hence I believe a lot of nicer things are possible like running a GA to get the good launch parameters automatically, finding an explicit Poincare Map and doing eigenvalue analysis and finally hopping over uneven terrain. There is a lot of scope for further work in terms of bifurcation analysis, path following, hopping with minimal expenditure of energy etc.

3. **Mechanical** Another major task was to fabricate the robot. The robot is being fabricated and should be ready by this week . There was a lot of delay in getting the robot fabricated partly due to the hunt for a better design and partly due to the complexity of the mechanisms involved.

   Electronics The electronics has been designed and is ready. Essential tasks such as event detection, motor control and sensor filtering were duly concentrated upon to ensure enough computing power for the control law. Wireless debugging was extensively used to analyse codes.

4. Experimentation with the actual system was a crucial aspect of the project which couldn't be completed. It will need another few weeks to get the robot up and running. I aim to convert the controller in embedded form and demonstrate a running gait as further work.

Appendix A

Equations of motion

```
xl[t_]= x[t] + (d - (l[t]- h) Tan [theta[t]]) Cos [theta[t]];
yl[t_]= x[t] + (l[t] - h) Cos[theta[t]] + d Sin[theta[t]];

xw[t_]= x[t] - (dw -d) Cos[theta[t]];
yw[t_]= y[t] -(dw -d) Sin[theta[t]];

xp[t_]= x[t] + d Cos[theta[t]];
yp[t_]= y[t] + d Sin[theta[t]];

T = 1/2 mw (xw'[t]^2 + yw'[t]^2) +  1/2 mp (xp'[t]^2 + yp'[t]^2) +  1/2 ml (xl'[t]^2 + yl'[t]^2) +
1/2 Jw (phi'[t] + theta'[t])^2 + 1/2 Jb theta'[t]^2;
V = g (ml yl[t] + mw yw[t] + mp yp[t]) + 1/2 K (l[t] - l0)^2;

L = T - V;
eq1 = D[D[L, D[x[t], t]], t]== D[L, x[t]];
eq2 = D[D[L, D[y[t], t]], t]== D[L, y[t]];
eq3 = D[D[l[t], t],t] == uL[t];
eq4 = D[D[L, D[theta[t], t]], t] == D[L, theta[t]];
eq5 = D[D[L, D[phi[t], t]], t] == D[L, phi[t]];
eqFlight = {eq1, eq2, eq3, eq4, eq5} // Simplify;

xfootOld = x0 + ((len - len0) Sin[theta0] + d Cos[theta0]);
eqConstraintStanceY = y ->  Function[t, (len - l[t])*Cos[theta[t]] - d*Sin[theta[t]]];
eqConstraintStanceX = x -> Function[t, xfootOld - ((len - l[t]) Sin[theta[t]] + d Cos[theta[t]])];
eq1 = D[D[L, D[l[t], t]], t] == D[L, l[t]] /. eqConstraintStanceX /. eqConstraintStanceY;
eq2 = D[D[L, D[theta[t], t]], t] == D[L, theta[t]] /. eqConstraintStanceX /. eqConstraintStanceY;
eq3 = D[D[L, D[phi[t], t]], t] == D[L, phi[t]] /. eqConstraintStanceX /. eqConstraintStanceY;
eqStance = {eq1, eq2, eq3} // Simplify;

eqConstraintSpring = l -> Function[t, len0];
eq1 = D[D[L, D[x[t], t]], t] == D[L, x[t]] /. eqConstraintSpring;
eq2 = D[D[L, D[y[t], t]], t] == D[L, y[t]] /. eqConstraintSpring;
eq3 = D[D[L, D[theta[t], t]], t] == D[L, theta[t]] /. eqConstraintSpring;
eq4 = D[D[L, D[phi[t], t]], t] == D[L, phi[t]] /. eqConstraintSpring;
eqSpring = {eq1, eq2, eq3, eq4} // Simplify;
```

Appendix B

Kalman Filter Implementation

```c
// Kalman filter variables
#define GYRO_SCALE              (0.07326)
#define ACCEL_SCALE             (0.0004625)
#define gravity                 (9.806)
#define FSAMP                   (50)
#define RAD2DEG                 (180.0/3.1415)

// 10.6
volatile INT16 GYRO_MULT = GYRO_SCALE*64.0;
volatile INT16 TILT_MULT = 0.1*64.0;

// Persistant states (10.6) -- covaraince is < 1
volatile INT16 P_00 = 64.0;
volatile INT16 P_01 = 0;
volatile INT16 P_10 = 0;
volatile INT16 P_11 = 64.0;

// Constants (10.6)
volatile INT16 A_01 = 64.0/FSAMP;
volatile INT16 B_00 = 64.0/FSAMP;

// Accelerometer variance (10.6) = 22*ACCEL_SCALE*g
volatile INT16 Sz = 22*ACCEL_SCALE*gravity*64.0;

// Gyro variance (10.6) = 96E-6
volatile INT16 Sw_00 = 1;

// Output (10.6)
volatile INT16 x_00 = 0;
volatile INT16 x_10= 0;

// Filter vars (all 10.6)
volatile INT16 inn_00 = 0, s_00 = 0, AP_00 = 0, AP_01 = 0, AP_10 = 0, AP_11 = 0, K_00 =
    0, K_10 = 0;
volatile INT16 to_send[7] = {0};

void do_kalman(INT16 gRead, INT16 tilt)
{
    // Update the state estimate by extrapolating current state estimate with input u.
    // x = A * x + B * u
    x_00 = x_00 + ((INT32)((INT32)A_01 * (INT32)x_10))/64 + ((INT32)((INT32)B_00 * (
        INT32)gRead))/64;
    to_send[0] = x_00;

    // Compute the innovation -- error between measured value and state.
    // inn = y - c * x
    inn_00 = tilt - x_00;
    to_send[1] = inn_00;

    // Compute the covariance of the innovation.
    // s = C * P * C' + Sz
    s_00 = P_00 + Sz;
    to_send[2] = s_00;

    // Compute AP matrix for use below.
    // AP = A * P
    AP_00 = P_00 + ((INT32)((INT32)A_01 * (INT32)P_10))/64;
    AP_01 = P_01 + ((INT32)((INT32)A_01 * (INT32)P_11))/64;
    AP_10 = P_10;
    AP_11 = P_11;
    to_send[3] = AP_00;

    // Compute the kalman gain matrix.
    // K = A * P * C' * inv(s)
    K_00 = ((INT32)((INT32)AP_00*64)) / s_00;
    K_10 = ((INT32)((INT32)AP_10*64)) / s_00;
    to_send[4] = K_00;

    // Update the state estimate
    // x = x + K * inn
    x_00 = x_00 + ((INT32)((INT32)K_00 * (INT32)inn_00))/64;
    x_10 = x_10 + ((INT32)((INT32)K_10 * (INT32)inn_00))/64;
    to_send[5] = x_00;

    // Compute the new covariance of the estimation error
    // P = A * P * A' - K * C * P * A' + Sw
    P_00 = AP_00 + ((INT32)((INT32)AP_01 * (INT32)A_01) - (INT32)((INT32)K_00 * (INT32)
        P_00))/64 + (INT32)(((INT32)((INT32)K_00 * (INT32)P_01)/64) * (INT32)A_01)/64 +
        Sw_00;
    P_01 = AP_01 - (INT32)((INT32)K_00 * (INT32)P_01)/64;
    P_10 = AP_10 + ((INT32)((INT32)AP_11 * (INT32)A_01) - (INT32)((INT32)K_10 * (INT32)
        P_00))/64 + (INT32)(((INT32)((INT32)K_10 * (INT32)P_01)/64) * (INT32)A_01)/64;
    P_11 = AP_11 - (INT32)((INT32)K_10 * (INT32)P_01)/64;
}
```

References

[1] A. Sayyad, B. Seth, and P. Seshu, "Single-legged hopping robotics research — A review," *Robotica*, vol. 25, no. 5, pp. 587–613, 2007.

[2] . http://www.ai.mit.edu/projects/leglab, Last checked on April 11th, "MIT Leg Lab," 2010.

[3] M. H. Raibert, "Legged robots," *Communications of ACM*, vol. 29, no. 6, pp. 499–514, 1986.

[4] W. Lee and M. Raibert, "Control of hoof rolling in an articulated leg," *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pp. 1386–1391, 1991.

[5] G. Zeglin, "Uniroo: A One-Legged Dynamic Hopping Robot," tech. rep., Massachusetts Institute of Technology, 1991.

[6] P. Gregorio, "Design, Control and Energy Minimization Strategies for the ARL Monopod," Master's thesis, McGill University, 1994.

[7] M. Ahmadi and M. Buehler, "Preliminary Experiments with an Actively Tuned Passive Dynamic Running Robot," *Experimental Robotics V*, 1997.

[8] J.-C. Zufferey, "First Jumps of the 3D Bow Leg Hopper," tech. rep., The Robotics Institute, Carnegie Mellon University, 2001.

[9] M. Ahmadi and M. Buehler, "A control strategy for stable passive running," *Proc. IEEE/RSJ Conf. on Intelligent Robots and Systems*, vol. 3, pp. 152–157, 1995.

[10] P. Shanmuganathan, *Dynamics and Stabilization of Under-Actuated Monopedal Hopping.* PhD thesis, Indian Institute of Technology Bombay, Dept. of Mechanical Engineering, July 2002.

[11] A. Sayyad, *Dynamics and Control of a Single Legged Offset Mass Hopping System.* PhD thesis, Indian Institute of Technology Bombay, Dept. of Mechanical Engineering, 2007.

[12] V. Saboo, "Development of a Hopping Robot," tech. rep., Indian Institute of Technology, Bombay, 2008.

[13] S. Pradhan, "Prototype Development for Hopping Height Control of a One-Legged Hopping Robot," tech. rep., Indian Institute of Technology, Bombay, 2009.

[14] S. Issani, "Design and Implementation of Reaction Wheel Mechanism to Control a One-legged Hoppng Robot," tech. rep., Indian Institute of Technology, Bombay, 2009.

[15] H. Goldstein, C. Poole, and J. Safko, *Classical Mechanics.* Addison Wesley, 2000.

[16] R. M. Ghigliazza, R. Altendorfer, P. Holmes, and D. Koditschek, "A Simply Stabilized Running Model," *Society for Industrial and Applied Mathematics*, pp. 519–549, 2005.

[17] D. Kirk, *Optimal Control Theory - An Introduction.* Dover Publications, 1998.

[18] J. Hespanha, D. Liberzon, and A. Teel, "Lyapnov Conditions for Input to State Stability of Impulsive Systems," *Automatica*, 2008.

[19] . http://renaissance.ucsd.edu/CoordinatedRoboticsLab/iHop.html, Last checked on April 12th, "iHop at UC San Diego's Co-ordinated Robotics Lab," 2010.

[20] S. P. Bhat, "Controllability of Non-Linear Systems," tech. rep., 2007.

[21] P. Chaudhari, "Design and Stabilization of a One Legged Hopping Robot - Stage I," tech. rep., Indian Institute of Technology Bombay, 2009.

[22] B. Seth, P. Seshu, P. V. Shanmuganathan, V. V. Vichare, and P. Raj, "Search for Initial Conditions for sustained hopping of Passive Springy Leg Offset Mass Hopping Robot," tech. rep., 2007.

Acknowledgment

I would like to thank Prof. Seth for the constant guidance and insights he has given me during the entire duration of this project. I am grateful to him for willing to let me work on my dream project at a very short notice. Prof. Arya's inputs have been invaluable to ensure that I think systematically about the different aspects of this problem. A number of ideas related to design as well as analysis wouldn't have been possible without the brainstorming sessions that we regularly have.

I would also like to thank Siraj and Simit for giving me access to all of their previous work. This immensely accelerated the development of the micro-controller based components of this project. The directions in which their projects have focussed have been very useful to me for deciding my own course.

Declaration of Integrity

I declare that this report represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea / data / fact / source in my report. I understand that any violation of the above will be cause for disciplinary action as per the rules and regulations of the institute.

Pratik Chaudhari

April 15, 2010