SYSTEM DYNAMICS-BASED MAPPING FOR CLOSED LOOP CONTROL

A Thesis
Presented to
The Academic Faculty

by

Anushri Dixit

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Science in the
School of Electrical Engineering with the
Research Option in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
MAY 2017

SYSTEM DYNAMICS-BASED MAPPING FOR CLOSED LOOP CONTROL

Approved by:

Dr. Patricio Vela, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Erik Verriest
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: 4th May, 2017

To Dr. Patricio Vela, of Georgia Institute of Technology.

ACKNOWLEDGEMENTS

I would like to thank my mentor and professor, Dr. Patricio Vela, for giving me this incredible opportunity. His enthusiasm about research has been ever so contagious. He has been a fantastic mentor and has set an example for what it means to be a professional in academia.

I wish to also extend my gratitude to my Ph.D. mentor, Alex Chang, for his unwavering patience and belief in my abilities. His dedication to research has served as an inspiration to me.

Lastly, I would like to thank my parents and my sister who have always loved and supported me.

TABLE OF CONTENTS

ABSTRACT

This paper focuses on obstacle avoidance using the walking gait and vision. Inspiration for the gaits was drawn from behaviors found in nature as well as prior contributions in the field of robotic locomotion. A quadrupedal robotic platform was designed and fabricated to support these studies and experiments. The walking gait was implemented on the platform using inverse kinematics and a map was developed connecting the system dynamics to the extrinsic control parameters, namely, the stride length of the robot and the turn of each leg. The paper has implications in path planning for bio-inspired robots in rough terrains. The goal of the research is the synthesis and evaluation of increasingly dynamic quadrupedal locomotion gaits like walking, trotting, and hopping for navigation over unknown terrain.

CHAPTER 1

INTRODUCTION

Biomorphism is the mirroring of natural patterns found in biological systems into man-made models. Biomorphic robots mimic the movement of animals so as to navigate over different kinds of terrain. Nature is often used as inspiration for building robots capable of locomotion; animals have evolved through natural selection to attain morphologies that enable them to competently navigate the topography of their native habitats. Incorporating these features into robots would contribute to the adoption of robotics in many application domains such as military and prosthetics. Regular wheeled vehicles face great difficulty in maneuvering through uneven and obstacle-ridden terrain. As their biological counterparts have demonstrated, legged robotic solutions have great potential to overcome this challenge. Quadrupeds, like cats and dogs, have more legs as compared to bipeds, like humans, and so intuitively they have more 'opportunity' or tools with which to maintain stability and perform locomotion.

A gait refers to the way animals move their limbs to move from one point to another. Various biomorphic robot platforms have been developed to mimic the different gaits, such as: walking, trotting, and hopping [1]. These gaits are then modified to traverse different kinds of terrains. Past research has often been focused on various aspects of improving the gaits that have been developed. This includes analyzing intrinsic factors and extrinsic factors affecting stability, development of design parameters for minimizing energy losses, and development of algorithms to traverse unknown terrain [1-6].

While some robots can trot at high speeds, others can overcome obstacles [4, 6]. The long-term goal of the research is to be able to combine different gaits such that the robot imitates quadrupedal animals to a much fuller extent. This paper takes a step in that direction by implementing closed loop control for maneuvering through an environment with obstacles. First, a simple walking gait is implemented using inverse kinematics. Next, the translational and rotational velocities of the robot are calculated for varying gait parameters. Finally, these velocities are used to obtain a path for traversing from one point to another. The path is calculated using non-linear optimization software, SNOPT.

CHAPTER 2

LITERATURE REVIEW

There are various components that go into creating a working quadruped robot; this section has been divided into four different components. While the parts given below may not cover all the different factors to be considered when one is developing a robot, for the purposes of this research these are the most important things to be considered. Ensuring stability was a crucial first step in gait generation. A lot of algorithms exist for generating a trajectory for the robot to travel from one point to another. Finding which method would be the best fit for this research required an analysis of the different algorithms and methods available. Further, different kinds of sensors can be used to emulate the response of a quadruped while traveling over uneven terrain. This section also aims to look at different ways in which robots have previously traversed unknown terrains and how mimicking animal responses has impacted gait generation.

Stability

There are two types of factors that affect the stability of a robot – intrinsic and extrinsic. Intrinsic factors include stiffness and damping at joints. Performance is usually measured in terms of locomotion stability, energetic efficiency, and physically reasonable gait criteria. Locomotion stability is sensitive to both knee stiffness and damping, and insensitive to hip impedance [2].

The center of mass and swing-foot are the extrinsic control parameters considered for the stability of the robot. A simulation model analyzes the performance of the robot based on the variation of these parameters. Three control methods for controlling the trajectory include single leg inverse kinematic control, whole body Jacobian control, and a hybrid control method [3].

Apart from the extrinsic and intrinsic factors, one should be able to minimize slippage at the point of contact between the foot and ground. Slippage is often hard to predict because surfaces properties vary, and it is also difficult to include the slippage in a simulation model.

Traversal over unknown terrain

One way to tackle traversal over unknown terrain is by restricting the gait to just one type, like step climbing. This is done by developing an algorithm through pre-planning of the steps to maintain stability throughout the gait. This algorithm is based on 3D sway compensation trajectory and helps position the center of gravity of the robot [7].

Traversal of unknown terrain is developed further using a best-search first algorithm. This algorithm is based on variations of foot placements, obstacle height, and body orientation. The performance improves when the robot looks ahead for more number of steps. However, it is never possible to be 100% sure that the obstacle can be traversed by the robot. Hence, there is a tradeoff between the number of steps that robot looks ahead to and the probability of actually overcoming the obstacle [5].

Trajectory Optimization

Trajectory optimization is finding a desirable set of solutions that satisfy a dynamic system with or without constraints. The dynamic system is usually defined using a set of differential equations.

There are various algorithms for trajectory optimization such as Newton's method, gradient descent, and optimal control. Specifically, gradient descent involves defining a cost function and moving along the path with the least slope (or gradient) to reach the best local solution that minimizes the cost. It is an effective way to find a solution to a nonlinear problem (NLP) with desired accuracy. However, gradient descent can only be implemented for unconstrained problems and is slow close to the minima [9].

The algorithms can be implemented for trajectory optimization by using shooting methods that solve a boundary value problem. Direct single shooting uses the initial conditions, final conditions, and the parameters of the system for defining the NLP variables. One drawback of single shooting is that small changes in the trajectory can easily affect the entire solution. To avoid this sensitivity, multiple shooting methods can be employed. In the case of multiple shooting, the trajectory is divided into smaller intervals. The multiple point boundary problems are then solved while ensuring continuity at the end of each boundary. As in the case of single shooting, direct multiple shooting also defines the initial conditions of each boundary value problem within the trajectory as the NLP variables [9].

Biologically-based control

Biologically inspired approach to gait generation uses a Central Pattern Generator (CPG) for locomotion control. CPG based control uses sensory input to change the period of its own active phase. The generator creates a pattern for the trajectories and uses feedback to modify these trajectories (response). Further, if sensor feedback is available, joint torque generation is implemented. This feedback is called a reflex. Biologically inspired control is done through responses and reflexes [1, 10].

It is important to acknowledge that it is difficult to completely emulate biological counterparts in robotics. To be able to mimic natural responses would require a lot of sensors that may not be completely necessary for the gait that is to be implemented. Hence, one must be able to decide to what extent the robot should be inspired biologically. After a point, it becomes costly to create a stable robot with a lot of sensor feedback.

CHAPTER 3

MATERIALS AND METHODS

This section outlines the methods used for gait generation. It starts with how the quadruped robot was designed and the development of its walking gaits. The design is important to understand because these design specifics result in the system dynamics calculated for obstacle avoidance. Then the generation of the walking gait through inverse kinematics is demonstrated. The walking gait is modified to generate a map of the velocities to the control parameters of the robot. Lastly, visual feedback is used to sense the environment to support closed loop control. Throughout, the code is first tested in a MATLAB simulation and then on the robot itself.

Development of a Quadruped Robotic Platform

A quadruped robotic system was fabricated to serve as our experimental platform. It was motivated by the open design of the miniature MIT Cheetah, a derivative of the larger MIT Junior. 3D printing was used to create the proper physical structure. However, the major difference was that instead of having a point contact at the end of the leg, a curved base was used. The curved base enabled a line contact with the ground for all robot configurations of interest, making the legs, and hence the body, more stable. Control and actuation was accomplished using Robotis Dynamixel servo motors (AX-18 and AX-12) and a Robotis OpenCM micro controller. The robot seemed to slip quite a bit on smooth surfaces and on uneven surfaces. This slip made it more unbalanced and prevented it from walking in a straight line. Adding rubber pads to the curved base helped reduce the slip and that

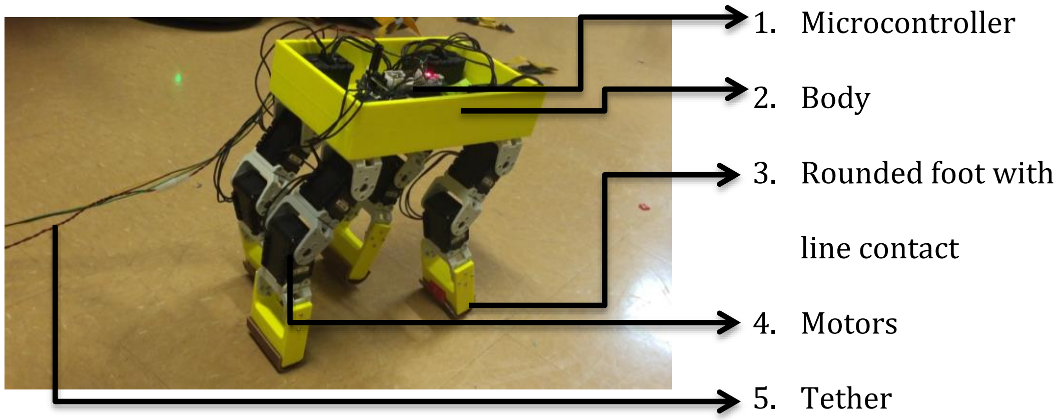


Figure 1. Different components of the quadrupedal system

significantly improved its performance, see Figure 1.

Walking Gait Generation using Inverse Kinematics

The walking gait has been designed such that the body of the robot moves forward with a constant velocity. The following variables have to be pre-defined (Figure 2):

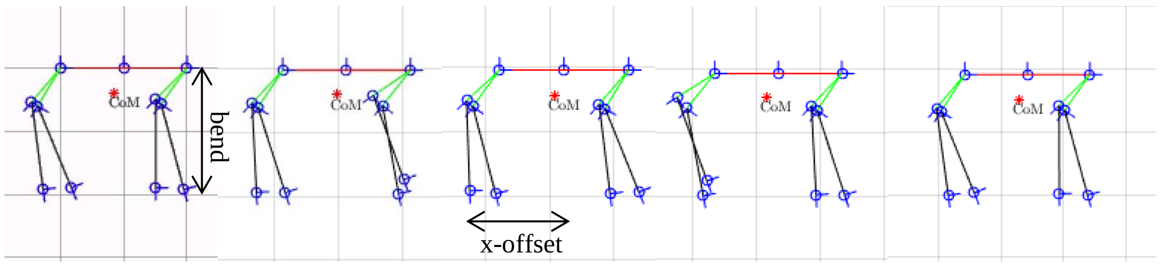


Figure 2. Trajectory of the feet in the walking gait

v: The constant velocity of the body of the robot (velocity of the Center of Mass).

T: The time taken for all the four legs to step forward (gait time period).

x_offset: Lift-off x-position of the foot with respect to the body of the robot.

bend: Lift-off y-position of the foot with respect to the body of the robot.

If the body has to move forward with a velocity v and only one leg steps forward at a time, it follows that the other three legs, that have to be stationary with respect to ground, move with a velocity –v. Furthermore, if each leg moves back with a velocity v for time 3T/4, it has to step forward with velocity 3v for time T/4 so that there is no overall change in configuration.

When deciding which *(x_offset, bend)* values were suitable to be used for the gait, a sweep of x-positions from a negative value (where we know the foot can't reach) to a positive value (where, once again, we know the foot can't reach) was done.

Figure 3 shows the inverse kinematic joint solutions (in degrees) when the leg moves with a velocity of 3v for T/4 of the time and with a velocity of -v for 3T/4 time (x-position). For the first T/4 time range, the leg steps forward in a sine curve trajectory (y-position).

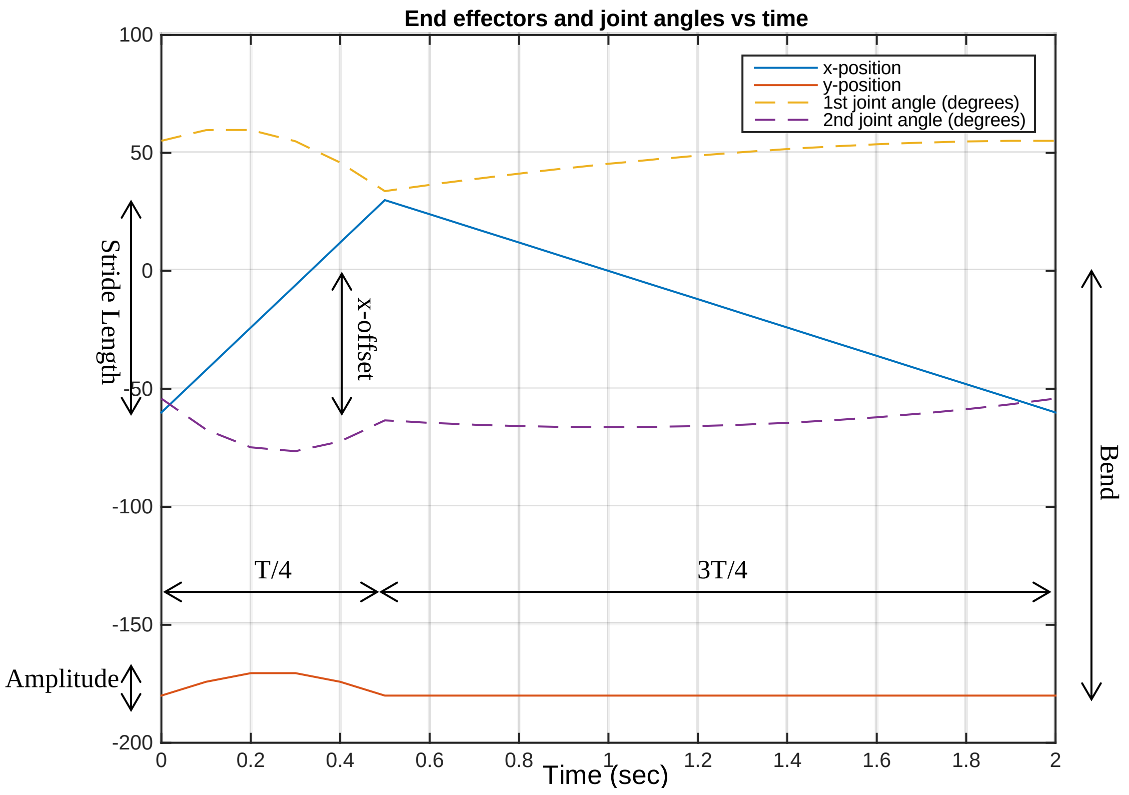


Figure 3. End effectors and joint angles vs. time for straight line walking

Steps for tracking using an overhead Kinect

The previous section implemented walking in a straight line. The legs were not angled and the robot velocity was constant. When the front legs are angled by some angle, the robot walks in a curved, circular path. Hence, it has translational and rotational velocities, also known as twist, that change based on how much the legs are turned. The twist also changes based on how much distance each foot covers when stepping forward. This distance is the stride length, see Figure 3. The stride length and the turn angle are control parameters of the walking gait.

The relationship between the twist and the control parameters was calculated through vision feedback. This feedback used a Kinect, the overheadTracker package provided by the Intelligent Vision and Automation Laboratory (IVALab), and the RANSAC Toolbox by Marco Zuliani [12]. The following steps outline how to extract the twist through visual sensing:

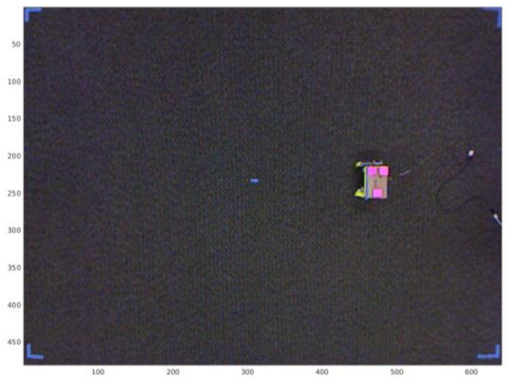
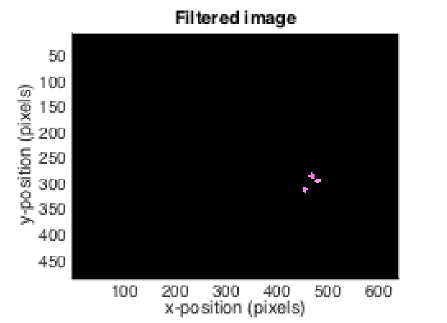


Figure 4. Overhead capture of the robot with distinct markers

1. Record the videos of the robot moving in circles wherein trajectory radii vary with leg turning angles (0, …, 45 degrees) and the stride lengths (70, 80, …, 110mm).
2. Overhead visual sensing of planar environment through 3 distinct markers on the robot, see Figure 4. The color composition of the magenta tags was 'trained' into a filter. This RGB color filter was then applied to recorded video data to track marker centroids, see Figure 5. The three marker-arrangement allows extraction of orientation.

3. Use the RANSAC algorithm to approximate the radius and center of the circle traversed.
4. Calculate the rotational velocity of the robot about the center of the circle (no translational velocity component). First, find the projection of each point on the robot trajectory on the approximated circle. Then, find the change in angle every few seconds. The average of this change gives the overall rotational velocity of the robot, see `Figure 6`.
5. Use this spatial twist to calculate the body twists of the robot, see `Figure 6`.
6. Interpolate the relationship between leg rotation angles, stride length, and body twist.



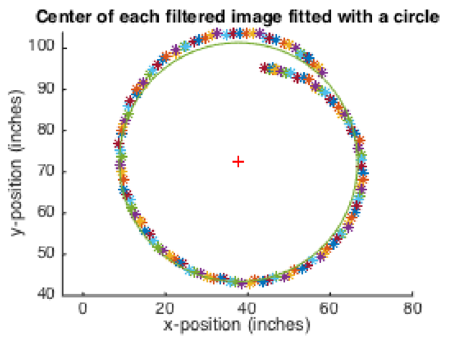
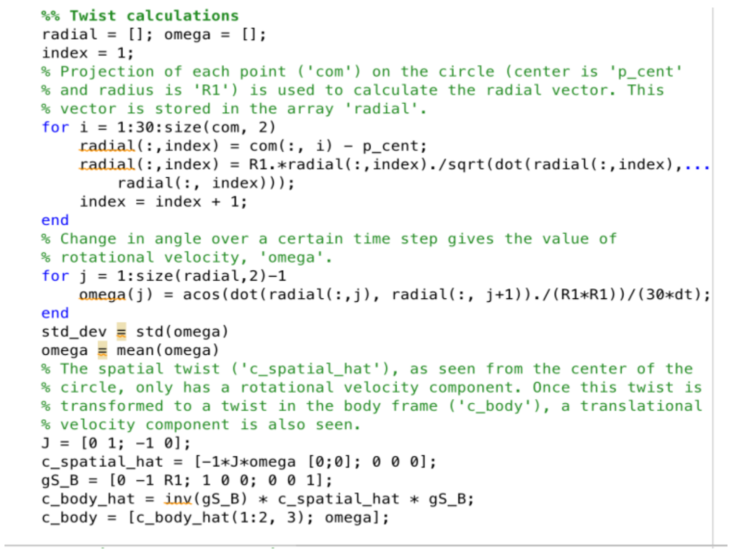


Figure 5. Filtered markers and the overall trajectory of the robot. The RANSAC circle fit is shown in the lower figure.



```
%% Twist calculations
radial = []; omega = [];
index = 1;
% Projection of each point ('com') on the circle (center is 'p_cent'
% and radius is 'R1') is used to calculate the radial vector. This
% vector is stored in the array 'radial'.
for i = 1:30:size(com, 2)
    radial(:,index) = com(:, i) - p_cent;
    radial(:,index) = R1.*radial(:,index)./sqrt(dot(radial(:,index),...
        radial(:, index)));
    index = index + 1;
end
% Change in angle over a certain time step gives the value of
% rotational velocity, 'omega'.
for j = 1:size(radial,2)-1
    omega(j) = acos(dot(radial(:,j), radial(:, j+1))./(R1*R1))/(30*dt);
end
std_dev = std(omega)
omega = mean(omega)
% The spatial twist ('c_spatial_hat'), as seen from the center of the
% circle, only has a rotational velocity component. Once this twist is
% transformed to a twist in the body frame ('c_body'), a translational
% velocity component is also seen.
J = [0 1; -1 0];
c_spatial_hat = [-1*J*omega [0;0]; 0 0 0];
gS_B = [0 -1 R1; 1 0 0; 0 0 1];
c_body_hat = inv(gS_B) * c_spatial_hat * gS_B;
c_body = [c_body_hat(1:2, 3); omega];
```

Figure 6. MATLAB code for steps 4 and 5 of tracking. The variable 'com' is an array of the x and y-positions of the robot at each point in its trajectory and 'p_cent' is the center of the circle obtained from the RANSAC approximation.

Obstacle Avoidance using Interpolated Data

Once the relationship between the control parameters and the body twist has been calculated, the desired velocity of the robot can be controlled by supplying the values of the control variables at any given time. Hence, the relationship can be used to generate a path that the robot can follow to travel from one point to another.

When formulating a path for traversal from one point to another, the main goal is to avoid any obstacles in between. Path generation for obstacle avoidance is done using Optragen [11]. Optragen is a 'user-convenience' framework that is built around an underlying optimization package, like IPOPT or SNOPT [13]. Specifically, Optragen converts an optimal control problem into a discrete set of problems to be solved as a nonlinear programming problem (NLP). This NLP is then solved using an optimization solver like SNOPT. Given below are the steps used for generating a trajectory for obstacle avoidance:

1. Position of center of mass of robot and its orientation is generated using control variables u (turn angle) and s (stride length) through the use of forward kinematics.
2. Use Optragen [11] to generate a trajectory for traveling from one point to another while avoiding given obstacles.

   This is done through appropriate constraints and the following assignments:

   Assignments:

   - *Nperiod*: Number of periods to run a control input before a switch
   - *ninterv:* Total number of input controls applied

- *T:* Time period of one gait cycle (typically 2 sec)

Constraints:

- *Initial value:* The initial x and y-positions have to be specified, along with the required orientation.
- *Final value:* Final x and y-positions, and the final orientation needed.
- *Controls:* The stride length (s) and turn angle (u) values are restricted. Further, the robot cannot turn with large turn angles when the stride length is too small. All of these constraints are given below:
  - $u \epsilon [-45°, 45°]$
  - $s \epsilon [70, 110]\ mm$
  - $s - |u| \geq 45$
- Obstacles: The position of the obstacles to be avoided needs to be expressed mathematically.
- Overall trajectory: The robot needs to follow the dynamics map found through the tracking. The overall velocities need to be expressed in terms of the controls.

Closed Loop Control

The previous section discussed how to generate a set of values for the control variables such that the robot avoids obstacles while walking. However, when these values are actually implemented on the physical system, the path the robot follows doesn't necessarily coincide with the path that generated by Optragen. Closed loop control is implemented to correct the errors in the path followed by the robot.

At any time instance, Optragen generates a twist that is to be applied to the system. This twist component is called feed forward. For closed loop control, there is another component to the twist applied, the feed back component. This component is calculated by finding the error in expected and actual position and orientation of the robot. An overhead Kinect gives feedback about the actual position of the robot. Hence, the following variables can be defined:

- g_{des}: Expected configuration of the robot based on trajectory found,
- g_{cur}: The actual current configuration of the robot as measured by the Kinect,
- g_{err}: The error between the expected and actual configurations of the robot

$$g_{err} = g_{cur}^{-1} * g_{des}$$

- c_{ff}: The feed forward component of the twist, calculated using the 'u' and 's' values generated by Optragen,
- c_{fb}: The feed back component of the twist

$$c_{fb} = \begin{bmatrix} K_x * x_{err} \\ 0 \\ K_y * y_{err} + K_\theta * \theta_{err} \end{bmatrix}$$

- c_{cmd}: The overall twist applied, in the body frame

$$c_{cmd} = c_{ff} + c_{fb}$$

Note: All configurations, g's, have (x, y, θ) components.

A forward mapping, describing the values of the control variables, u and s, for different twist values was created. In order to create this map, a gradient descent function was written to solve for the controls when provided with a twist. The gradient descent function was then called for various values of the twist, uniformly spread over the entire measured range, and the controls calculated were stored in the forward map. The idea was to calculate the error (g_{err}) between the desired position (g_{des}) and the actual position (g_{cur}) using an overhead Kinect and to then correct this error using proportional control. The values of u and s were then interpolated for the new twist (c_{cmd}) from the forward map.

CHAPTER 4

RESULTS

The walking gait

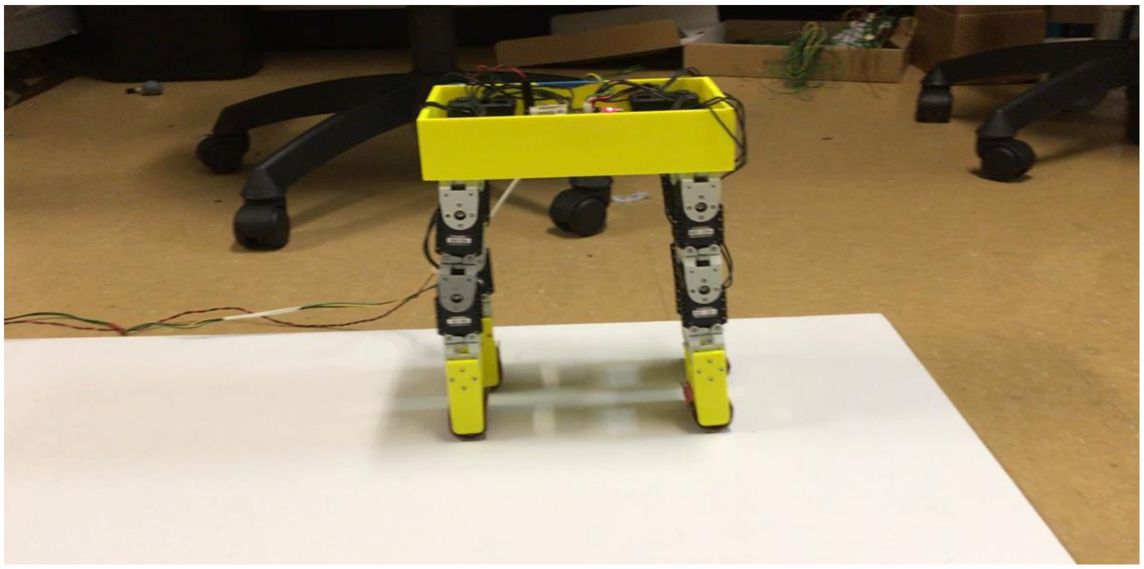


Figure 7. Video of the walking gait (Link to video)

System dynamics-based mapping

System dynamics govern how the robot responds to control inputs. In this case, an empirical model of the walking kinematics was developed. The body twist, consisting of the translational (Figure 9) and rotational (Figure 8) velocities, was measured for different control inputs, stride length and turn angle.

*Rotational Velocity (*ω*):*

We expected the rotational velocity to be a function of turn angle only, independent of the stride length. The final relationship obtained, through a surface fit to the data collected using the procedure outlined earlier, was consistent with this hypothesis.

$$\omega = 1.872 * 10^{-6} * u^3 - 4.019 * 10^{-10} * u^2 - 0.009778 * u$$

Where, u is the turn angle is measured in degrees and ω is the rotational velocity measured in rad/sec. It should be noted that since u is in degrees, the powers of u in the equation contribute to a significant change in values of ω despite the coefficients being really small, especially since the range of ω is [-0.27, 0.27] rad/sec.

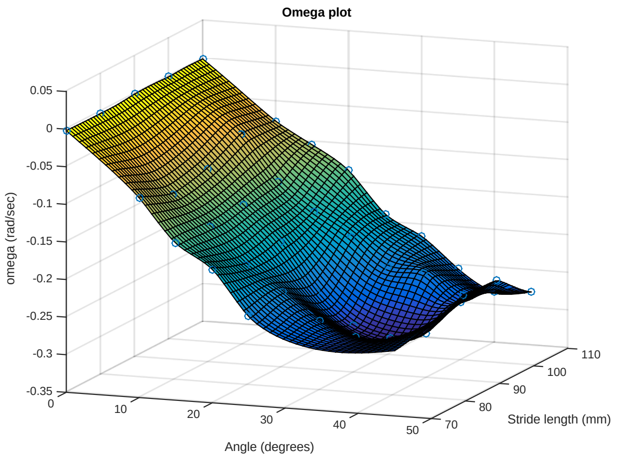


Figure 8. Plot of rotational velocity (rad/sec) vs. turn angle (degrees) and stride length (mm)

Translational Velocity (v):

We expected the translational velocity to be a function of stride length only, independent of the turn angle. More specifically, we anticipated the velocity to be directly related to the stride length. However, the final relationship was not in agreement with what was expected. The value of v decreased with the increase in |u| and s.

$$v = 6.522 * 10^{-16} * u - 0.006479 * s - 1.357 * 10^{-17} * s * u - 0.0005495 * u^2 + 4.579$$

Where, u is the turn angle is measured in degrees, s is the stride length measured in mm and, v is the translational velocity in inches/sec. Once again, the coefficients are very small. In this case, however, the range of v is [2.12, 4.48] inches/sec. Hence, smaller changes in v from terms like $6.522 * 10^{-16} * u$ can be ignored. Despite this assumption, v is still dependent on both u and s.

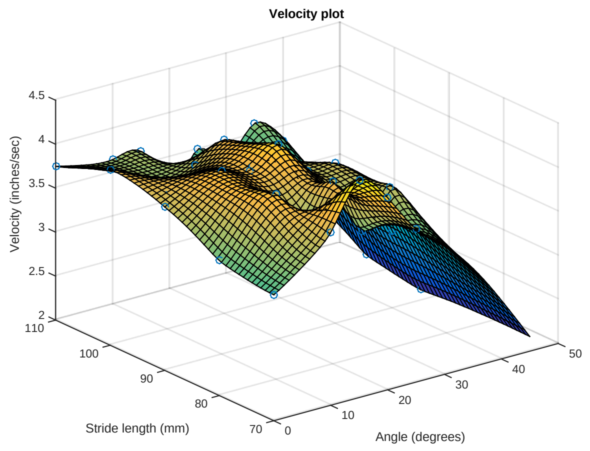


Figure 9. Plot of translational velocity (inches/sec) vs. turn angle (degrees) and stride length (mm)

Path planning using Optragen

Using the relationships found through the mapping, Optragen constraints were defined. The following plots (Figure 10) show the path taken by the robot and its orientation at each instance for the example considered in the methods section earlier. Note that the small red circles in the XY position plot denote the obstacles that the robot is supposed to avoid.

In this case,

- *Nperiod* = 4
- *ninterv* = 7
- *(x, y, θ)$_i$* = (0, 0, 0)
- *(x, y, θ)$_f$* = (-50, 50, pi/2)
- $Total\ time = Nperiod * ninterv * T$

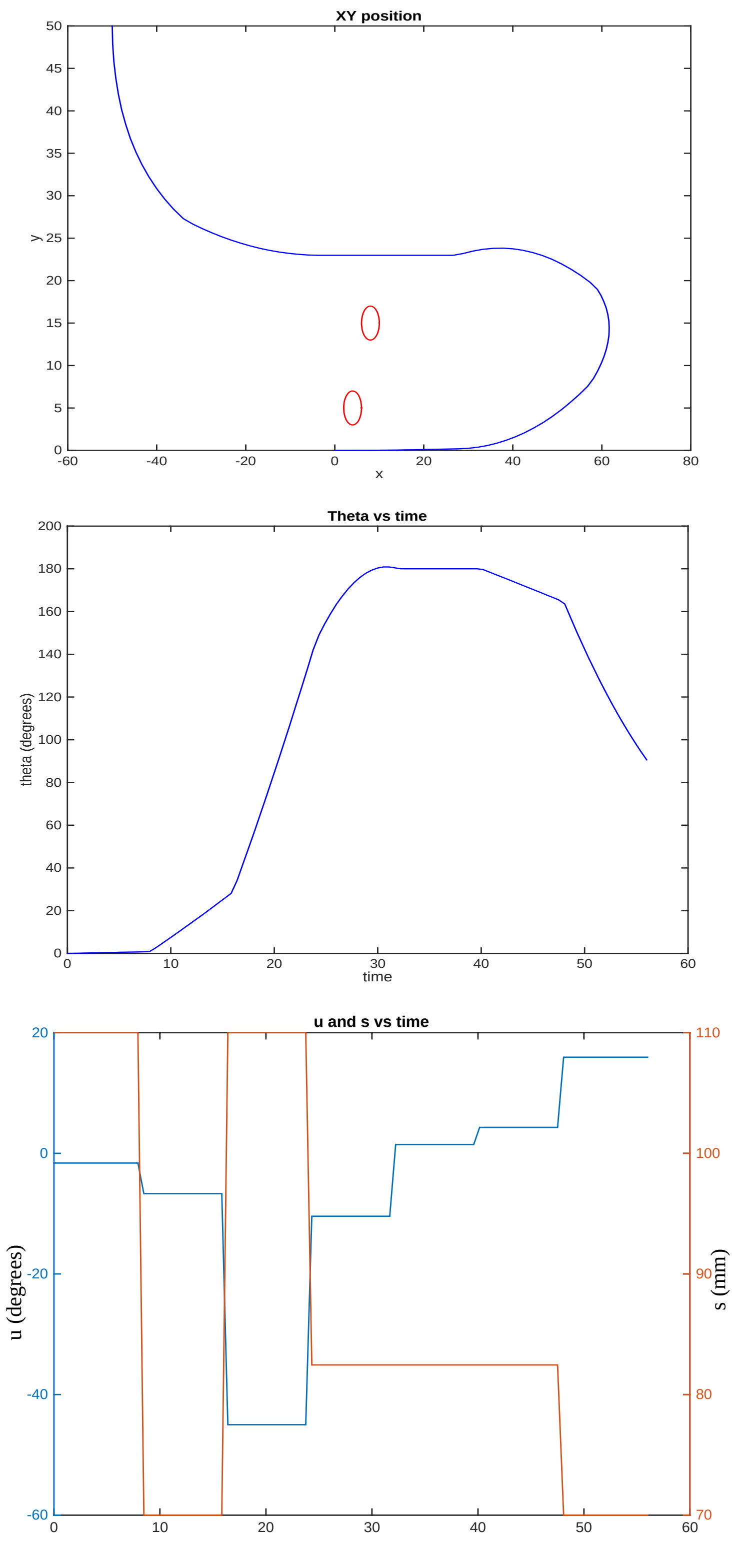
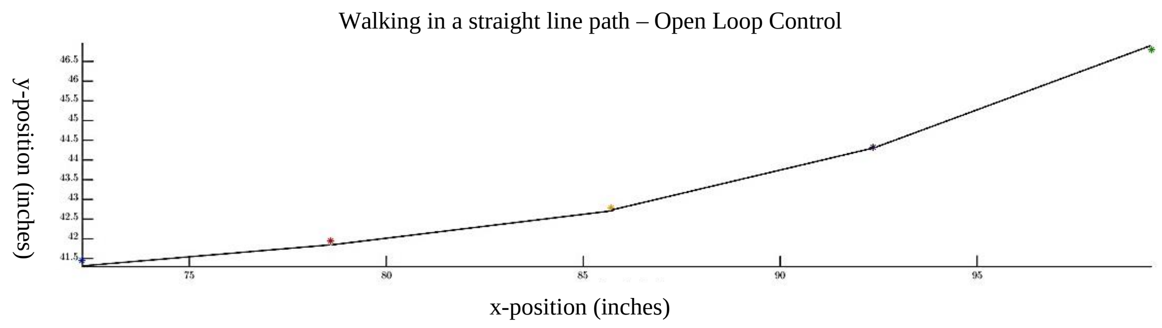


Figure 10. Plots of the x-y position of the robot, its overall orientation, and the controls to be commanded as it avoids the obstacles in its path

Closed loop control for a Straight-Line path

Closed loop control was initially tested for a simpler example of traveling in a straight-line path before implementing the same to obstacle avoidance that usually resulted in a curved path. This testing was done using the same feedback control formulas outlined in the methods section. For this case, $K_x = 0.1$, $K_y = -0.01$, $K_\theta = -0.05$.

Figure 11, shows the accuracy in open loop control vs. that in closed loop control. This proportional control is then applied to the trajectory that is generated for obstacle avoidance so that the path followed by the robot is more accurate.



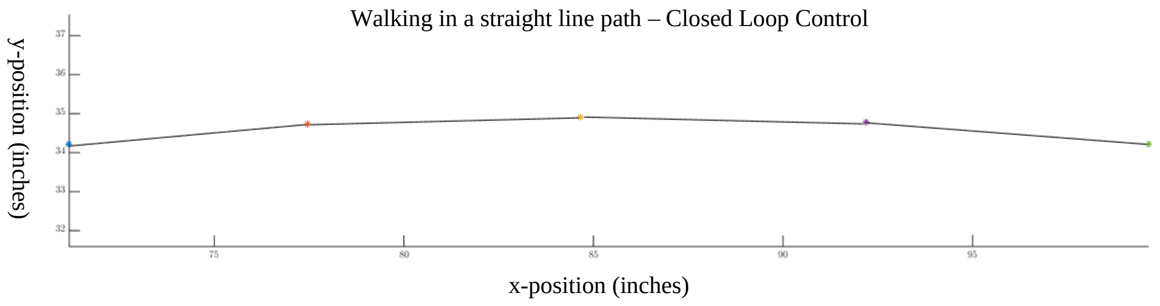


Figure 11. Open loop vs. closed loop control for a straight-line path

Closed loop control for Obstacle Avoidance

After a trajectory is calculated using Optragen, it is applied to the physical system. With open loop control, the system does not follow the desired path with good accuracy. The results of closed loop proportional control are closer to the desired path. After every few seconds, the error between the desired and actual position is calculated and the twist applied is changed accordingly.

Figure 12 shows two trajectories:

- The green one is the desired path, calculated using Optragen,
- The red trajectory is the actual path followed by the robot using closed loop control.

The red circles indicate the obstacles in the path.

The initial configuration was expected to be (0,0,0) and the final configuration was supposed to be (90, 70, pi/2). Note that the robot does not start exactly at (0,0,0) and instead is slightly off that position initially. It continuously lags behind the desired trajectory and hence is not able to reach the final configuration.

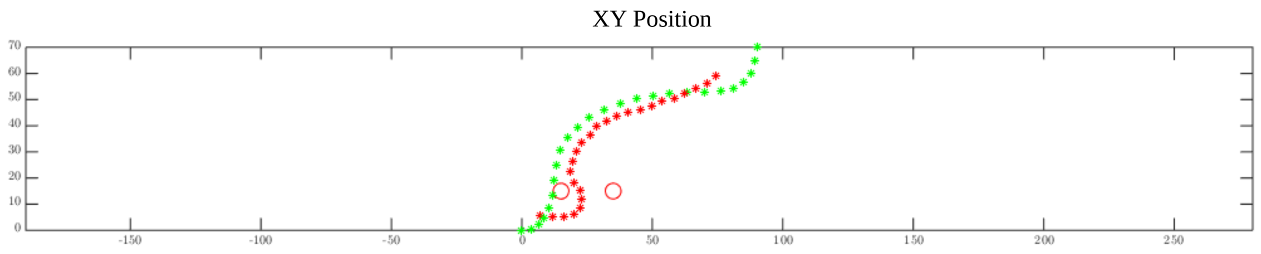


Figure 12. Desired vs. actual trajectory in closed loop control

CHAPTER 5

DISCUSSION

The mapping of control parameters (u, s) to the body twist shows that the rotational velocity is independent of the stride length but that the translational velocity is dependent on both stride length and the turn angle. These relationships only hold for particular ranges of u and s.

- $u \epsilon [-45°, 45°]$
- $s \epsilon [70, 110]\ mm$
- $s - |u| \geq 45$

The last relationship is a mathematical restriction reflecting the observation that the walking gait is extremely unstable for instances when the value of stride length is small and the value of the turn angle is large. For example – the robot tips over for any turn angle over 25° when the stride length is 70mm. This happens because the stability of the robot outside of the region covered by the inequality is very poor. As the legs turn more and more, the stability polygon gets smaller and smaller.

Under these conditions, the translational and rotational velocity cannot be controlled independently as was expected earlier. A nonlinear relationship was obtained between the twist and the control variables. The system constraints are solved using a nonlinear problem solver, SNOPT that is used by Optragen. Optragen solves for the desired path using gradient descent. The different gaits can be combined into one non-differentiable (discontinuous) function. Because gradient descent requires a continuous function, multiple gaits, like the walking gait and the turn in place gait, cannot be implemented at the same time as was expected for obstacle avoidance over rough terrain. The inability to combine different gaits is the reason why we need to use the multiple shooting method instead of the single shooting method that was in use. In the case of multiple shooting, the overall desired path could be divided into smaller paths over multiple time steps while ensuring continuous boundary conditions between the different time intervals. Now, for each time interval, a different gait can be implemented.

The closed loop control with feedback from the overhead Kinect corrects errors in navigation. The turn angle is calculated from the rotational velocity equation first and then the best stride length that matches the needed translational velocity is computed through gradient descent. Greater emphasis is placed on the rotational velocity while path planning because, right now, it matters more that the robot correctly follows the path provided as opposed to traveling along the path quicker. If we wanted the opposite to be true, translational velocity would be given more emphasis. If the rotational and translational velocity were independent of each other, that is, if the translational velocity had been independent of the turn angle, as was expected earlier, it could've been ensured that equal emphasis be placed on both translational and rotational velocity.

In addition to the walking gait, trotting and turn in place gaits were also implemented. These gaits were a simple variation of the walking gait. The trotting gait involved two legs, diagonally opposite each other, stepping forward at a time. The turn in place gait used the trotting gait, with the front legs turned by 90° and the back legs turned by -90°.

The goal of this research was to evaluate different kinds of gaits and how they can be implemented for traversal over unknown terrain. The results obtained in this study have been a step towards that. In the future, the aim would be to combine different gaits available into the path-planning algorithm through the method of multiple shooting. Further, newer gaits that will be more conducive to traversing over rough terrain need to be developed. Overall, the system needs to be made more dynamic.

REFERENCES

[1] B. Thomas, E. Alexander, T. Arndt von, and B. Ansgar, "Controlling legs for locomotion—insights from robotics and neurobiology," *Bioinspiration & Biomimetics,* vol. 10, p. 041001, 2015.

[2] W. Bosworth, K. Sangbae, and N. Hogan, "The effect of leg impedance on stability and efficiency in quadrupedal trotting," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014)*, 2014, pp. 4895-4900.

[3] A. Shkolnik and R. Tedrake, "Inverse Kinematics for a Point-Foot Quadruped Robot with Dynamic Redundancy Resolution," in *2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 4331-4336.

[4] S. Sangok, A. Wang, C. Meng Yee, D. Otten, J. Lang, and K. Sangbae, "Design principles for highly efficient quadrupeds and implementation on the MIT Cheetah robot," in *2013 IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 3307-3312.

[5] B. Mitchell, A. G. Hofmann, and B. C. Williams, "Search-based Foot Placement for Quadrupedal Traversal of Challenging Terrain," in *2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 1461-1466.

[6] D. Pongas, M. Mistry, and S. Schaal, "A Robust Quadruped Walking Gait for Traversing Rough Terrain," in *2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 1474-1479.

[7] S. Kitano, S. Hirose, G. Endo, and K. Suzumori, "Trot gait based feed-forward walking on challenging terrain: Case of high step climbing," in *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2015, pp. 2519-2524.

[8] M. Raibert, "Hopping in legged systems - Modeling and simulation for the two-dimensional one-legged case," *IEEE Transactions on Systems, Man and Cybernetics,* vol. SMC-14, pp. 451-463, 1984.

[9] J. T. Betts, "Survey of numerical methods for trajectory optimization", *Journal of Guidance. Control, and Dynamics.* vol. 21, no. 2, 1998.

[10] Y. Fukuoka, H. Kimura, and A. H. Cohen, "Adaptive dynamic walking of a quadruped robot on irregular terrain based on biological concepts," *International Journal of Robotics Research,* vol. 22, pp. 187-202, 2003.

[11] R. Bhattacharya, "OPTRAGEN: A MATLAB Toolbox for Optimal Trajectory Generation," in *Proceedings of the 45th IEEE Conference on Decision and Control*, San Diego, CA, 2006, pp. 6832-6836.

[12] M. Zuliani, "RANSAC Toolbox for Matlab", 2008.

[13] P. E. Gill, W. Murray, M. A. Saunders, "SNOPT: An SQP algorithm for large-scale constrained optimization", *SIAM Rev.*, vol. 47, no. 1, pp. 99-132, Mar. 2005.