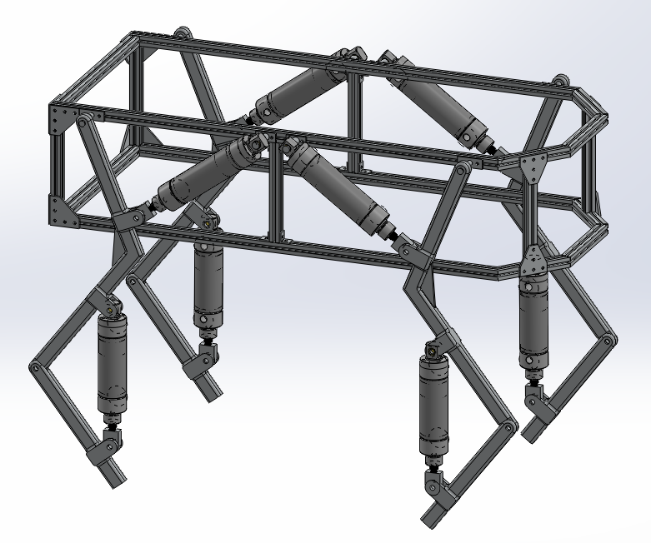
http://www.joyglobal.com/clientCSS/images/logo_JoyGlobal.png

**Development of an Agile Educational Robot**

Team A.R.C. – Design Report



**February 23, 2015**

Logan Beaver

Justin Campbell

Tyler Paddock

Ronald Shipman

***Advisor***: Dr. Luis A. Rodriguez

Contents

[Table of Figures 3](#_Toc412410254)

[Table of Tables 3](#_Toc412410255)

[Executive Summary 4](#_Toc412410256)

[Dynamics 5](#_Toc412410257)

[Mechanical Design 14](#_Toc412410258)

[Pneumatic Component Specifications 15](#_Toc412410259)

[Electrical System 15](#_Toc412410260)

[State Machine and Control 16](#_Toc412410261)

[Appendix I – Dynamic Simulation MATLAB Code 18](#_Toc412410262)

# Table of Figures

[Figure 1: Mathematical notation for the robot model. 5](file:///D:\MyDocs\Documents\GitHub\AgileRoboticControls\Documentation\Design%20Report\Design%20Report%20Agile%20Quadruped%20Project.docx#_Toc412410029)

[Figure 2: Free body diagrams for the shank (left) and thigh (right).. 6](#_Toc412410030)

[Figure 3: Free body diagram of the chassis at an arbitrary angle. 6](#_Toc412410031)

[Figure 4: Length definitions for the front right robot leg 8](file:///D:\MyDocs\Documents\GitHub\AgileRoboticControls\Documentation\Design%20Report\Design%20Report%20Agile%20Quadruped%20Project.docx#_Toc412410032)

[Figure 5: Kinematic step simulation for a single robot leg 10](#_Toc412410033)

[Figure 6: Free body diagram for the drag portion of the gait 11](#_Toc412410034)

[Figure 7: PID model for a single leg 16](#_Toc412410035)

[Figure 8: Flowchart representation of the state machine architecture. 17](#_Toc412410036)

# Table of Tables

[Table 1: Results for the dynamic walking simulation 12](#_Toc412410154)

# Executive Summary

The Milwaukee School of Engineering (MSOE) participates in Science Technology Engineering and Mathematics (STEM) outreach events for prospective students. The school will benefit greatly from having a sophisticated robotic control system to build excitement about STEM as well as sparking interest in fluid power, automation, and the controls fields. An agile pneumatic robot is not only a complicated control system that can be used to get young people excited about STEM, but it will also increase the prestige of MSOE knowing that a group of seniors attending the school were able to design and build the system from the ground up. In addition it also provides an exciting opportunity for future groups to iterate on the design and integrate new and exciting features.

To fulfill the needs of the project existing robot designs were researched to help determine the initial objectives and constraints for the project. Existing walking robots such as Boston Dynamics Big Dog and Little Dog, the Swiss Federal Institute of Technology (EPFL) Cheetah Cub, and various robots from the Massachusetts Institute of Technology Computer Science and Artificial Intelligence (CSAIL) laboratory were examined. These robots were used as a baseline comparison for the design specifications and constraints. From the robots the following constraints and project goals were identified for this project’s design.

* A maximum weight of 35 Kg for portability
* Robot fitting within a 0.75 m x 0.75 m x 1.0 m box for portability
* Custom debug panel creation to facilitate controls troubleshooting
* MATLAB and Simulink model support to allow mechanical engineering students to update control algorithms without knowledge of C/C++
* Electronic fuses and shielding to protect the robot and operator during use and maintenance
* Mechanical protection to reduce the risk of pinching and self-collision damage to the robot
* An easy to access emergency stop to quickly depower the robot
* A pressure relief valve to reduce the risk of overloading and damaging pneumatic components

The work done on this project is a continuation of the work done by Kevin Lee during the Research Experience for Undergraduates (REU) at MSOE. His work involved deriving a dynamic model for a simplified quadruped robot. This work is continued by the agile robotics controls team in deriving a full dynamic model for the physical robot and integrating it with control algorithms to manipulate the robot. This resulting robot design will be implemented in actual hardware toward the end of the project.

Pneumatic power was chosen over electronic and hydraulic power for a variety of reasons. Pneumatics were chosen over hydraulics due to the weight and maintenance needs associated with hydraulic systems. Hydraulic systems are also dirtier than pneumatic systems and pneumatic working fluid is freely available. Pneumatics were chosen over electronic systems due to their higher power density. Electrical systems have lower power density from the inefficiencies in converting electrical energy to work. In addition fluid power systems are compliant, meaning that if a large force is applied to the pneumatic actuators the fluid can compress whereas electronic actuators will experience an increased stress.

The robot locomotion utilizes a quadruped design. Four legs were selected because of the inherent static stability of a four legged design coupled with the decreased control complexity compared to robots with additional legs over four. This will allow the robot to initially actuate a slow statically stable gait as the software architecture is developed and will eventually lead to more sophisticated gaits being developed without the need for additional hardware.

The controls implementation will initially be done using a software simulation. This allows rapid updating of the main codebase as the mechanical and electronic designs iterate. Eventually the controls will be implemented into a main microcontroller which will take user input through a human machine interface (HMI) and relay the commands to the pneumatic actuators.

Four design alternatives were drafted to fulfill the design requirements. The design alternatives were named *Arachnia*, *Hexabox*, *Boxxy*, and *DogeBot*. After scoring each robot with a design matrix, *DogeBot* was chosen as the design to be continued, with a score of **96.19** out of 100.

# Dynamics

To determine the internal forces felt in the joints and the required torques for locomotion a dynamic mathematical model of the robot was constructed. From the specifications the robot was known to have four legs with two links each all attached to a main chassis. Summing the force and torque around each link of the robot resulted in 27 simultaneous equations used to calculate the state of the robot. To simplify the calculations it was assumed that the robot exhibited purely planar motion and all leg torque is applied purely at the hip and knee. A diagram of the mathematical notation used in the model is shown below:

X

Y

Θthigh

Θbody

Θshank

Leg 1

Leg 2

Leg 3

Leg 4

Thigh

Shank

Figure : Mathematical notation for the robot model. The body angle is measured relative to the horizontal, the thigh angle is measured relative to the body, and the shank angle is measured relative to the thigh.

To determine the equations of motion for the overall system free body diagrams were developed for the shanks, thighs, and bodies. This resulted in a system of 27 equations with 32 unknowns. The free body diagrams are given below for a single leg and the chassis:

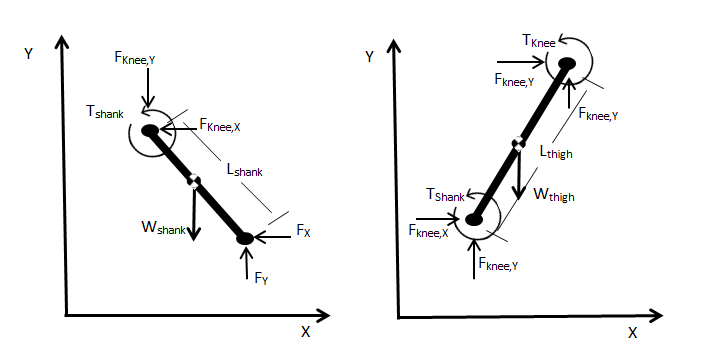


Figure : Free body diagrams for the shank (left) and thigh (right). Summing forces and masses on the legs generates 24 equations and 32 unknowns.

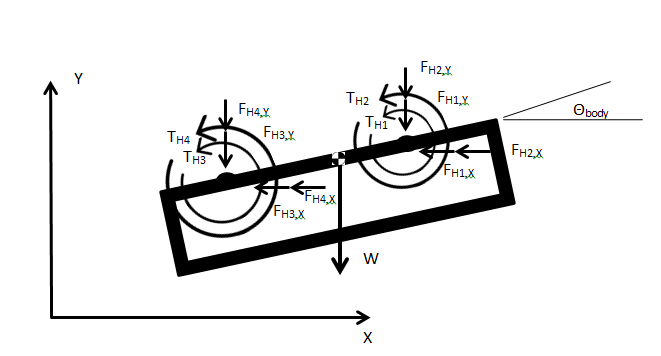


Figure : Free body diagram of the chassis at an arbitrary angle. The body brings the system to 27 equations and 32 unknowns.

To simplify the model into a solvable form it was assumed that the reactionary forces of the feet were known, and equations of motion for the body were discarded. This reduced the simulation to a system of 24 equations and unknowns, which could be solved to find the torque applied and internal force felt at each joint.

The final step in developing a solvable model for the robot is to deal with the non-inertial reference frame of the robot. Due to the robot’s acceleration during motion, taking a force and moment sum of each joint will neglect inertial forces acting on the robot. The coriolis and centrifugal forces are two examples of inertial forces that need to be accounted for when doing a force sum in a non-inertial reference frame. **Reference here**.

There are three main ways to deal with dynamics in non-inertial reference frames. The most common way is to simply add in the inertial forces to make newton’s laws of motion valid. Adding in the inertial forces is the most conceptually straight forward and the simplest to solve mathematically. However without a physical system to test it would be very intensive to determine the extent that non inertial forces are affecting a system as complicated as ours.

The second method is to use lagrangian mechanics to solve the system in an arbitrary coordinate system. Due to the general energy based nature of lagrangian mechanics inertial forces are accounted for during equation derivation. Unfortunately the core curriculum at MSOE does not mention lagrangian mechanics and it would be too time intensive to learn an additional dynamic system on top of constructing the senior design project. **Refrences here**.

The final method is to have an arbitrary ground point as the reference for the system. This eliminates the inertial forces acting on the robot while also allowing newton’s laws of motion to be used for the dynamical analysis. After giving the robot an arbitrary ground location the sum of the torque equation needed to be modified using the **parallel axis theorem?** A subset of the developed equations is given below:

Where *r* is a distance vector to the arbitrary ground point, *Fhip* and *Fknee* are the internal forces on the hip and knee joints, *Thip* and *Tknee* are the torques applied to the hip and knee joints, and *Tequivalent* is calculated as follows:

Where *Izz* is the moment of inertia of the link about the z axis. **SITE SOURCE PLZ**.

The torque and force equations can then be put into the following matrix forms:

Where T is an 8x1 matrix of torques, A is an 8x8 matrix of 1’s and 0’s, and B is an 8x1matrix containing the other torque equation elements such as distances and equivalent torques. F is a 16x1 matrix of internal joint forces, C is a 16x16 matrix of 1’s and 0’s, and D is a 16x1 matrix containing masses, accelerations, and weights.

Equations ### can then be solved by inverting the A and C matrices to get the following solution form:

This creates a system of equations where the torques and forces are dependent on the robot link’s mass and inertia, the angular position, velocity, and acceleration of each joint, and the Cartesian acceleration of each joint. The mass and inertial values for the robot were taken from the SolidWorks model developed during the previous design phase and a kinematic model was developed to calculate the required position, velocity, and acceleration values during motion.

To determine the position of the hip, knee, foot, and thigh CGs a kinematic model for the robot was developed. The kinematic model assumed each leg is a serial manipulator grounded at the CG of the robot at a position (x, y) relative to the arbitrary ground. A diagram for the front right leg is given below:

LH1

LK1

LF1

LT1

LS1

Figure : Length definitions for the front right robot leg. All positions are measured with polar notation using a series of angles and distances from the point (x,y) at the CG of the robot chassis. For the kinematic equations the leg is considered a serial manipulator consisting of revolute joints connected to a ground reference at point (x, y). All angles use the earlier notation.

From the above diagram it is easy to perform a serial manipulator analysis to determine the position of any point on the robot leg. The position of the foot in terms of the body, hip, and knee angles is given below:

Where F is the Cartesian foot position. All angles follow the convention given earlier.

The equations were then symbolically stored in a MATLAB script and derived with the following generic function **reference**:

Which is equivalent to:

Where *J(f)* is the Jacobian matrix of function *f*. This approach allows rapid iteration of design versions without changing the differentiation of the joint velocity and accelerations. The partial derivative of the position is multiplied by the position Jacobian to acquire the velocity function, and the velocity function is multiplied by the velocity Jacobian to achieve the acceleration. A simplified form of the kinematics which assumed *LH1 =* θ*body = 0* was hand derived and matched exactly with the MATLAB differentiation.

In addition a static case close form solution was created for the torques and forces at each joint in a single leg. After modifying the simulation parameters to fit a single static leg the identical equations were output by the simulation. Although having a correct static case doesn’t verify the entire dynamical derivation, it is useful to spot any fundamental errors in the simulation.

After deriving the dynamic and kinematic equations the required torques and resultant forces at each joint are functions of the following 11 x 1 state vector q and its two time derivatives:

To calculate q and its time derivatives for the robot a second kinematic simulation was constructed with some simplifications. The simulation was simplified to a single leg with the foot following an arbitrary path at a constant speed. This allows the angles and their derivatives to be calculated for a single leg. The values of the other legs could then be approximated either as constant values or values based on from the simulation.

A semi-ellipse was selected as the initial foot path. The path was selected due to its mathematical simplicity and the ease of changing the step length and height parameters. It is also a similar shape to the more complicated pear-shaped quartic, which is the path many organic creature’s feet follow during motion. A constant speed was selected to simplify the step analysis. A plot of the simulation in action is given below:



Figure : Kinematic step simulation for a single robot leg. The leg follows the elliptical path with a constant velocity.

To get a close approximation of the maximum stresses and torques felt by the robot a 0.5 m long step was selected to be completed in 1 second which satisfies the specification of moving a maximum of 0.5 m/s.

After a close inspection of the step it should be clear that there is a corner at 0.25 m, which causes an unrealistically large spike in acceleration and therefore force and torques felt. To correct this error the step was split into three phases. The first two phases, swinging and dragging, were calculated as described above. The third phase, impulse, used the velocity of the leg to calculate the foot force which was then used to calculate internal forces and torques in the joints. The impulse method was derived as follows from the conversion of linear momentum **reference**:

Where *m* and *v* are the mass and velocity of their respective bodies, and *f(t)* is the impulse force applied to the foot. Assuming *f(t)* is constant and *vfloor* = *vtotal* = *0* the following equation results:

Which calculates the force applied to the foot. This force can then be used in conjunction with the other simulations to determine the impulse force felt by the robot joints.

After running the simulation for the swing and drag phase the vector *q* is also known for a single leg. In order to find the torques required and internal forces at each joint some simplifying assumptions were made about the robot’s gait to determine the remaining values in *q* and the foot forces.

The first simplification made was that the robot performed a drag gait to move. This means the robot locks three of its legs and uses the fourth to drag itself along the ground. It was also assumed that the robot chassis is moving at a constant velocity of 0.5 m/s forward. These simplifications don’t capture the entirety of the dynamical motion, but it is accurate enough to get a rough idea of the expected dynamic forces without spending months doing only dynamical analyses of the system.

To finish defining the state vector *q* a value of 90 degrees was selected for the remaining hip angles, and a value of zero degrees was selected for the body and knee angles. The derivatives of these values were also set to zero due to the static nature of this gait. Finally the foot force values were calculated for the swing and drag phases of the elliptical step. For the drag portion of the step the following free body diagram was used to calculate foot force values:

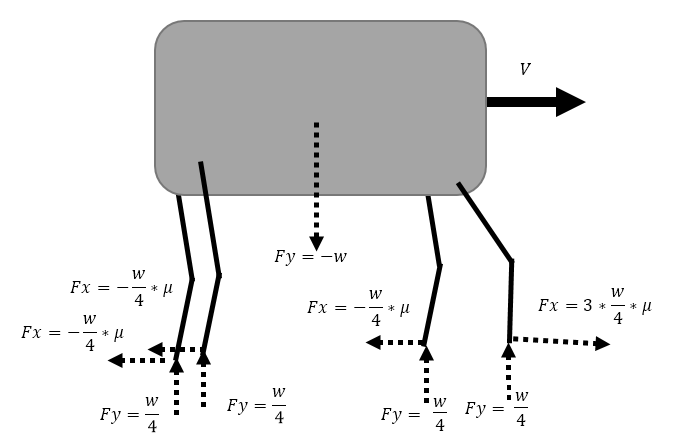


Figure : Free body diagram for the drag portion of the gait. All four legs are assumed to be on the ground of the robot while travelling at a constant velocity V.

Where μ is the dynamic friction coefficient of the feet. This resulted in the following equations:

Which, when simplified, leads to the following:

Equation **##** simplifies to weight divided by four because the rotational acceleration of the body was assumed to be zero. Thus to prevent an unbalanced tipping moment the force of each foot in the y direction must be equal. For the simulation a value of 0.5 was used for the coefficient of friction. This is a common value for rubber sliding across a hard surface. **Src?**

Finally the foot forces during the swing phase were calculated. To simplify the calculations it was assumed that the robot body was completely static as the foot swung out. There are a few inaccuracies in this assumption, but the amount of time it would take to determine a foot reaction force model for the robot would exceed the time allotted for the project. With this simplification and leg geometry setup it is trivial to calculate that the x and y components of force acting on the swinging leg is zero. Additionally because the legs are completely vertical it is trivial to conclude the x component of the other foot forces are zero as well, while the y direction is one third of the robot weight.

After performing all of the derivations a MATLAB script was developed to run the simulations and output the state of the robot at the maximum torque and maximum internal forces. The script can be seen in Appendix 1. The maximum values output from the script are tabulated in the figure below:

Table : Simulation results for the Swing, Drag, and Impulse phases of the simulation. Values are labeled as [Type][Location][Leg], so TK1 is the Torque applied at the Knee on Leg 1, and FH2x is the Force felt by at the Hip on Leg 2 in the X direction.

|  |  |  |  |
| --- | --- | --- | --- |
| *Value* | *Swing* | *Drag* | *Impulse* |
| TH1 [N-m] | 383.2008 | 67.082 | 0 |
| TH2 [N-m] | 2.9008 | -3.1367 | 0 |
| TH3 [N-m] | -0.7823 | -1.3573 | 0 |
| TH4 [N-m] | 2.9008 | -3.1367 | 0 |
| TK1 [N-m] | 1.3135 | 99.3253 | 0 |
| TK2 [N-m] | 17.5289 | 6.8914 | 0 |
| TK3 [N-m] | -13.2918 | -9.8418 | 0 |
| TK4 [N-m] | 17.5289 | 6.8914 | 0 |
| FK1x [N] | 557.6 | 33.7092 | 12.3869 |
| FH1x [N] | 911 | 91.9408 | 0 |
| FK2x [N] | 0 | 17.25 | 0 |
| FH2x [N] | 0 | 17.25 | 0 |
| FK3x [N] | 0 | 17.25 | 0 |
| FH3x [N] | 0 | 17.25 | 0 |
| FK4x [N] | 0 | 17.25 | 0 |
| FH4x [N] | 0 | 17.25 | 0 |
| FK1y [N] | 789.9 | 145.8055 | 50.3962 |
| FH1y [N] | 1321.2 | 199.8142 | 34.5 |
| FK2y [N] | -42.4 | 30.8703 | 34.5 |
| FH2y [N] | -38.2 | 26.652 | 34.5 |
| FK3y [N] | -42.4 | 30.8703 | 34.5 |
| FH3y [N] | -38.2 | 26.652 | 34.5 |
| FK4y [N] | -42.4 | 30.8703 | 34.5 |
| FH4y [N] | -38.2 | 26.652 | 34.5 |

It can be seen from the above table that the forces and torques are an order of magnitude higher during the swing phase compared to the other two. At first this may seem unintuitive, but it does make sense. The drag and impulse columns represent forces that are being used to either stop or maintain a velocity, whereas the swing column includes forces used to achieve high accelerations due to the motion of the foot.

With these simulation results the pneumatic cylinders and legs can be sized to fit the walking specifications. The results were used with a motion study and cost analysis to determine the maximum force output and therefore pressure required by the pneumatic system. This allowed the other components of the pneumatic circuit to be sized. Additionally the worst case scenario forces for the legs were determined allowing their design to be iterated and ensured they did not fail.

# Mechanical Design

The major mechanical components of the robot are the legs and the chassis. The legs are further broken down into the thigh and the shank. The constraints placed on the mechanical design include the overall dimensions, weight, and carrying capacity of the robot.

The chassis of the robot is designed to support the legs while in motion, as well as serve as the housing for the pneumatic and electrical components. The choice to use 6105 –T5 T-Slotted Aluminum framing for the chassis was due to the simplicity of construction. With a number of connection plates and brackets available, a simple chassis would be easy to construct. In addition to its simplicity, the Aluminum framing is lightweight, fulfilling the robot’s weight constraint, and strong enough to support the forces exerted by the pneumatics during operation. Custom hip joints will need to be designed to be able to easily attach the legs to the outside of the frame. Additionally, mounts for the pistons will need to be created in a similar fashion.

The design of the legs is based on the anatomy of quadruped mammals, more specifically dogs. In order to reduce the robots complexity, the number of joints in the legs was reduced from three to two. In order to compensate for the loss in range of motion, the thigh was designed with a bend in it. The bend allows for a shorter stroke length for the pneumatic cylinder. This in turn, means that the cylinder can be attached closer to the hip joint on both the body and the thigh. The bend in the thigh is also beneficial in that it prevents the piston controlling the rotation of the knee joint from reaching a singularity point and possibly seizing. The design of the shank is much simpler. One factor that helped to determine the lengths of the thigh and shank was the desired step length. Similar to the body, the legs will most likely be constructed using an Aluminum alloy. A specific material is yet to be determined and will be based on the required strength and machinability of the design.

As a means to ensure the design would be handle the forces due to the weight and pistons, an initial FEA analysis was run. Each component was tested separately, using the worst case scenario as an upper limit test of the structures: Trying to move at maximum velocity using the slowest gait with the joints seized. Under these conditions, the components hold up, meaning the designs should be more than capable of handling normal operating conditions.

An important design consideration focused on in this phase of the project was the design of the robot’s feet. The foot will provide the necessary friction required to prevent the robot from slipping on its walking surface. Options for feet of the robot include rubber, either spheres or rectangular sleeves, in which the lower shank is inserted and liquid rubber that can be applied to the bottom of the shank and allowed to cure. The rubber sphere feet allow the foot to contact the ground at any orientation, while still providing the necessary friction, however if the shank cannot be securely inserted, the effectiveness of the foot during the leg’s motion becomes a major concern. Rubber sleeves face the same considerations as rubber spheres in that the shank must be inserted securely, so for both options methods for ensuring a sound connection must be looked into. If appropriate sizes of the rubber spheres or sleeves are not available, custom making the appropriate feet is then required. When considering the need of custom manufacturing the feet of the robot, ultimately the use of a liquid rubber substance, such as urethane, that can be cured into the necessary shape, as an option for the feet of the robot becomes the most attractive. By cutting a hole into the bottom of the shank and dipping the shank into the liquid rubber, the appropriate shape of the leg is formed and, as liquid rubber flows through the hole in the shank and cures, a connection point between the foot and the shank is formed. By repeating this dipping process, an appropriate layer of rubber can be applied to the lower portion of the shank, generating an effective foot for the robot. Initial testing of the chosen liquid rubber substance will be conducted to confirm desired mechanical properties.

# Pneumatic Component Specifications

After the dynamic simulation of the robot’s legs was conducted and the necessary forces required to generate the torques in the legs were determined, initial air cylinder specifications could be made. These initial specifications were iterated based on appropriate bore sizes, related to leg dimensions, pressures required in cylinders to produce necessary forces, utilizing Equation 1 and the necessary volumetric flow rates of the cylinders, utilizing Equation 2, to produce the required forces. These values can then be used to specify compressor pressure and storage required. If the pressures necessary in the air cylinders was too high based on the achievable output pressure of the air compressor, a larger bore diameter was chosen to lower this necessary pressure. If the required volumetric flow rate of the cylinders was too large, options to lower this includes reducing the stroke length, decreasing the bore diameter of the cylinder, or reducing the required pressure in the air cylinder. After specifying air cylinders and the air compressor, the components that need to be specified include the relieve valve and the directional control valves that allows the necessary volumetric flow to the air cylinders.

(1)

where F is the required force to generate the required torque at one of the joints of the robot, Acap is the area of the cap end of the piston and Arod is the area of the rod end of the piston.

(2)

where Q is the required volumetric flow rate, Ap is the surface area differential of the cap end and rod end of the piston, DT is the total displacement of the piston in 1 cycle, N is the number of cycles per minute and C is the compression ratio comparing the working pressure in the air cylinder to atmospheric pressure. A Sample calculation of cylinder pressure and necessary volumetric flow rate can be found at the end of the appendix.

# Electrical System

The electronics of the robot are broken up into two major subsystems, the motherboard and the debug panel. The motherboard was designed to contain the auxiliary electronics and signal conditioning components needed for the robot. The debug panel contains all necessary electronics to display battery levels and other statuses of the robot.

The motherboard contains the signal conditioning for each of the pneumatics cylinders. Each pneumatic cylinder is controlled by a signal analog direct current voltage. However, the signal driving this analog voltage is a pulse-width-modulated (PWM) output on the microcontroller. To convert a PWM into an analog signal an active low pass filter is used. After the low pass filter an opto-isolator is used to separate the microcontroller circuit from the pneumatic actuator circuit. An opto-isolator works by converting an electrical signal into an optical signal by using a diode. The optical signal is recaptures within the device and output onto another circuit as a current signal. At the output of the opto-isolator a trans-impedance amplifier is used to convert the output current signal to a voltage signal for the solenoid of the pneumatic valve. To handle the feedback signal from the pneumatic actuator another opto-isolator is used to separate the two power circuits then the signal is amplified before being read by the microcontroller’s built in analog to digital converters (ADCs).

The debug panel subsystem contains a physical panel with light emitting diodes (LED) and connections for banana plug cables. The LEDs are used to show battery levels and the status of the robot. The banana plug connectors are used to interface to Milwaukee School of Engineering’s test equipment in the labs. Banana plugs are used because they are standard on test equipment. A USB slot is also included on the debug panel to assist in programming the microcontroller while leaving it in the robot.

Two subcomponents of the debug panel are the 9 volt battery level indicator and the 24 volt battery level indicator. There were originally two options to make these indicators. The first option included using a LM 3914 chip, known as a dot/bar display driver. The LM 3914 uses 10 LEDs to create a dot graph or bar graph to display the magnitude of an input voltage. To implement the first option the battery voltage level would be connected to the LM 3914 and the 10 output LEDs used to display information to the robot operator. The second option to implement a battery level indicator uses zener diodes and LEDs. Three zener diodes with different threshold voltages are used to detect certain voltage levels of the battery. Depending on the voltage levels LEDs are turned on or off. To maintain the project timeline the simpler solution was chosen, which is using zener diodes and LEDs.

The 9 volt battery level indicator uses 3 zener diodes and LEDs. The LEDs are colored green, yellow, and red to indicate good voltage levels at green and bad voltage levels at red. To determine the voltage thresholds of the zener diodes the minimum voltage out of the battery to operate the robot was experimentally determined with an Agilent DC power supply. The microcontroller was connected to the DC power supply starting at 9 volts and slowly lowered until the microcontroller turned off. This lower voltage was measured at 4.5 volts. The following equations are used to pick the required zener diode threshold voltages. Vmax is the maximum voltage of the battery, Vmin is the minimum voltage the microcontroller can operate with. VT represented a voltage threshold step down from the maximum voltage. V x represents the zener diode threshold voltage of zener diode x, where x is Green, Yellow, or Red for the different color LEDs.

(Vmax – V­min)/3 = VT

VGreen = Vmax – (1) VT

VYellow = Vmax – (2) VT

VRed = Vmax – (3) VT

By solving the equations VGreen is determined to be 8.5 volts, VYellow is determined to be 6.0 volts, and VRed is determined to be 4.5 volts.

# Control Algorithms

The control algorithms are software implementations on the microcontroller. The software was not written in C or any text based language, but is instead code generated from Mathwork’s Simulink models. These Simulink models were generated by our team and are used to decrease the complexity of the programming. The control algorithm used is a proportional integral derivative controller using one input signal. This input signal is the position setpoint minus the feedback signal from the cylinder. Figure 1 shows the Simulink model for a whole leg using an upper and lower cylinder PID.

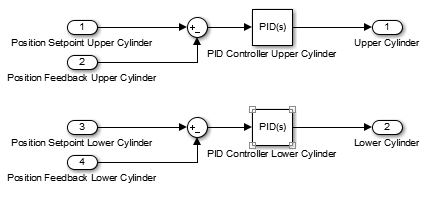


Figure : PID model for a single leg

# State Machine

It was determined that the behavior of the robot would be dictated utilizing a state machine. Using a state machine, based on the physical condition of the robot and the user input, the robot moves from different states and behaves occurring to how those states are defined within the state machine. Currently an initial state machine has been developed that has two states. The first state is a stand-by state in which the robot is determined to be functioning correctly and is awaiting user input. The second state is a stop state in which the robot, after some input which forces the robot into this state, completely stops ongoing motion in the robot and moves the robot into a stable position. Figure 8 shows a state machine flowchart in which various states are represented. States include forward and backward motion states in the robot, turning states and a stop state. The full state machine needs to be implemented in the next phase of the project for robot operation.



Figure : Flowchart representation of the state machine architecture.

# Appendix I – Dynamic Simulation MATLAB Code

%Handles configuration of the leg model and inputting values in LegModel.m

%Logan Beaver 10/23/14

clear; clc; close all;

Link1 = 'Thigh.stl';

Link2 = 'Shank.stl';

%read excel data%

CONSTANTS = ReadExcelData();

%assign the 12 read variables

totalWeight = CONSTANTS(1); % n

bodyLength = CONSTANTS(2); % m

bodyCg = CONSTANTS(3); % m from front

L1 = CONSTANTS(4); % cm1

iT = CONSTANTS(5); % m^4

thighCg = CONSTANTS(6); % m

tMass = CONSTANTS(7); % kg

L2 = CONSTANTS(8); % cm

iS = CONSTANTS(9); % m^4

shankCg = CONSTANTS(10); % m

sMass = CONSTANTS(11); % kg

frictionFactor = CONSTANTS(12); % unitless

%Cylinder velocities/accelerations, not relevant

L12 = L1; %cm

L21 = 0; %cm

P1x = -5; %cm

P1y = 0; %cm

P11 = L1/2; %cm

P21 = L1/2; %cm

P22 = L2/2; %cm

sampleDistance = 10; %cm - interpolation value for under-defined gaits

bodyVel = input('Insert Body Velocity In Meters per Second: '); % meters per second

stepDist = input('Insert Step Distance In Meters: '); % seconds

time = stepDist/bodyVel; %time to complete this step at the required speed

fprintf('Time Alloted for Step: %g seconds\n', time);

choice = input('Is this a foot drag (y/N): ', 's'); % seconds

if choice == 'y'

dragging = true;

fprintf('Running foot drag simulation...\n');

func = DragStep;

else

dragging = false;

fprintf('Running step simulation...\n');

func = SemiEllipseStep;

end

%read values from excel sheet

% ---- CONFIGURE STEP HERE BASED ON LEG LENGTHS AND STEP DISTANCE ------- %

Thetas = LegModel(Link1,Link2,L1,L2,L12,L21,P1x,P1y,P11,P21,P22,func,sampleDistance,time);

t = [0; cumsum(dt)];

theta1 = Thetas(:,1); %grabbing theta 1

theta2 = Thetas(:,2); %grabbing theta 2

dTheta1 = diff(theta1)./dt; %deltaTheta / deltaTime

dTheta2 = diff(theta2)./dt; %deltaTheta / deltaTime

ddTheta1 = diff(dTheta1)./dt(1:length(dt)-1);

ddTheta2 = diff(dTheta2)./dt(1:length(dt)-1);

MatrixDynamics();

%Put forces into moment equations

syms FH1x FH2x FH3x FH4x FH1y FH2y FH3y FH4y;

syms FK1x FK2x FK3x FK4x FK1y FK2y FK3y FK4y;

%Important values ---------------------------------- TOTAL WEIGHT

%subs forces into Torque equations

T = [Ts.TH1; Ts.TH2; Ts.TH3; Ts.TH4; Ts.TK1; Ts.TK2; Ts.TK3; Ts.TK4];

T = subs(T, FK1x, FORCES.FK1x);T = subs(T, FK2x, FORCES.FK2x);

T = subs(T, FK3x, FORCES.FK3x);T = subs(T, FK4x, FORCES.FK4x);

T = subs(T, FK1y, FORCES.FK1y);T = subs(T, FK2y, FORCES.FK2y);

T = subs(T, FK3y, FORCES.FK3y);T = subs(T, FK4y, FORCES.FK4y);

T = subs(T, FH1x, FORCES.FH1x);T = subs(T, FH2x, FORCES.FH2x);

T = subs(T, FH3x, FORCES.FH3x);T = subs(T, FH4x, FORCES.FH4x);

T = subs(T, FH1y, FORCES.FH1y);T = subs(T, FH2y, FORCES.FH2y);

T = subs(T, FH3y, FORCES.FH3y);T = subs(T, FH4y, FORCES.FH4y);

%create and subs forces

F = [FORCES.FK1x; FORCES.FH1x; FORCES.FK2x; FORCES.FH2x; ...

FORCES.FK3x; FORCES.FH3x; FORCES.FK4x; FORCES.FH4x; ...

FORCES.FK1y; FORCES.FH1y; FORCES.FK2y; FORCES.FH2y; ...

FORCES.FK3y; FORCES.FH3y; FORCES.FK4y; FORCES.FH4y];

%values to be set to constant values -------------------------SET ANGLES

syms x y tb th2 th3 th4 tk2 tk3 tk4;

syms xdot ydot tbdot th2dot th3dot th4dot tk2dot tk3dot tk4dot;

syms xddot yddot tbddot th2ddot th3ddot th4ddot tk2ddot tk3ddot tk4ddot;

%body vals---------------------------------------------------BODY ANGLES

T = subs(T, x, 0); T = subs(T, y, 0); T = subs(T, xdot, bodyVel); T = subs(T, ydot, 0);

T = subs(T, xddot, 0); T = subs(T, yddot, 0);

T = subs(T, tb, 0) ;T = subs(T, tbdot, 0); T = subs(T, tbddot, 0);

F = subs(F, x, 0); F = subs(F, y, 0); F = subs(F, xdot, bodyVel); F = subs(F, ydot, 0);

%forces

F = subs(F, xddot, 0); F = subs(F, yddot, 0);

F = subs(F, tb, 0) ;F = subs(F, tbdot, 0); F = subs(F, tbddot, 0);

%limb vals-------------------------------------------------3 LIMB ANGLES

T = subs(T, th2, pi); T = subs(T, th3, pi); T = subs(T, th4, pi);

T = subs(T, tk2, 0); T = subs(T, tk3, 0); T = subs(T, tk4, 0);

T = subs(T, th2dot, 0); T = subs(T, th3dot, 0); T = subs(T, th4dot, 0);

T = subs(T, tk2dot, 0); T = subs(T, tk3dot, 0); T = subs(T, tk4dot, 0);

T = subs(T, th2ddot, 0); T = subs(T, th3ddot, 0); T = subs(T, th4ddot, 0);

T = subs(T, tk2ddot, 0); T = subs(T, tk3ddot, 0); T = subs(T, tk4ddot, 0);

%forces

F = subs(F, th2, pi); F = subs(F, th3, pi); F = subs(F, th4, pi);

F = subs(F, tk2, 0); F = subs(F, tk3, 0); F = subs(F, tk4, 0);

F = subs(F, th2dot, 0); F = subs(F, th3dot, 0); F = subs(F, th4dot, 0);

F = subs(F, tk2dot, 0); F = subs(F, tk3dot, 0); F = subs(F, tk4dot, 0);

F = subs(F, th2ddot, 0); F = subs(F, th3ddot, 0); F = subs(F, th4ddot, 0);

F = subs(F, tk2ddot, 0); F = subs(F, tk3ddot, 0); F = subs(F, tk4ddot, 0);

%force values-----------------------------------------------FOOT FORCES

syms FF1x FF1y FF2x FF2y FF3x FF3y FF4x FF4y;

if dragging

T = subs(T, FF1x, 3\*totalWeight/4\*frictionFactor); T = subs(T, FF2x, totalWeight/4\*frictionFactor); T = subs(T, FF3x, totalWeight/4\*frictionFactor); T = subs(T, FF4x, totalWeight/4\*frictionFactor);

T = subs(T, FF1y, totalWeight/4); T = subs(T, FF2y, totalWeight/4); T = subs(T, FF3y, totalWeight/4); T = subs(T, FF4y, totalWeight/4);

%force

F = subs(F, FF1x, 3\*totalWeight/4\*frictionFactor); F = subs(F, FF2x, totalWeight/4\*frictionFactor); F = subs(F, FF3x, totalWeight/4\*frictionFactor); F = subs(F, FF4x, totalWeight/4\*frictionFactor);

F = subs(F, FF1y, totalWeight/4); F = subs(F, FF2y, totalWeight/4); F = subs(F, FF3y, totalWeight/4); F = subs(F, FF4y, totalWeight/4);

else

T = subs(T, FF1x, 0); T = subs(T, FF2x, 0); T = subs(T, FF3x, 0); T = subs(T, FF4x, 0);

T = subs(T, FF1y, 0); T = subs(T, FF2y, totalWeight/3); T = subs(T, FF3y, totalWeight/3); T = subs(T, FF4y, totalWeight/3);

%force

F = subs(F, FF1x, 0); F = subs(F, FF2x, 0); F = subs(F, FF3x, 0); F = subs(F, FF4x, 0);

F = subs(F, FF1y, 0); F = subs(F, FF2y, totalWeight/3); F = subs(F, FF3y, totalWeight/3); F = subs(F, FF4y, totalWeight/3);

end

%masses and weights ---------------------------------------MASSES

syms mS1 mS2 mS3 mS4 mT1 mT2 mT3 mT4 g;

T = subs(T, mS1, sMass);T = subs(T, mS2, sMass);T = subs(T, mS3, sMass);T = subs(T, mS4, sMass);

T = subs(T, mT1, tMass);T = subs(T, mT2, tMass);T = subs(T, mT3, tMass);T = subs(T, mT4, tMass);

T = subs(T, g, 9.81);

%forces

F = subs(F, mS1, sMass);F = subs(F, mS2, sMass);F = subs(F, mS3, sMass);F = subs(F, mS4, sMass);

F = subs(F, mT1, tMass);F = subs(F, mT2, tMass);F = subs(F, mT3, tMass);F = subs(F, mT4, tMass);

F = subs(F, g, 9.81);

%Lengths--------------------------------------------------LIMB CGs

syms LH1 LH2 LH3 LH4 LK1 LK2 LK3 LK4 LF1 LF2 LF3 LF4 LT1 LT2 LT3 LT4 LS1 LS2 LS3 LS4;

%L1 L2 L3 L4 are converted to m from cm by dividing by 100

T = subs(T, LK1, L1/100);T = subs(T, LK2, bodyLength-L1/100);T = subs(T, LK3, L1/100);T = subs(T, LK4, bodyLength-L1/100);

T = subs(T, LF1, L2/100);T = subs(T, LF2, L2/100);T = subs(T, LF3, L2/100);T = subs(T, LF4, L2/100);

T = subs(T, LH1, bodyCg);T = subs(T, LH2, bodyCg);T = subs(T, LH3, bodyCg);T = subs(T, LH4, bodyCg);

T = subs(T, LT1, thighCg);T = subs(T, LT2, thighCg);T = subs(T, LT3, thighCg);T = subs(T, LT4, thighCg);

T = subs(T, LS1, shankCg);T = subs(T, LS2, shankCg);T = subs(T, LS3, shankCg);T = subs(T, LS4, shankCg);

%forces

F = subs(F, LK1, L1/100);F = subs(F, LK2, bodyLength-L1/100);F = subs(F, LK3, L1/100);F = subs(F, LK4, bodyLength-L1/100);

F = subs(F, LF1, L2/100);F = subs(F, LF2, L2/100);F = subs(F, LF3, L2/100);F = subs(F, LF4, L2/100);

F = subs(F, LH1, bodyCg);F = subs(F, LH2, bodyCg);F = subs(F, LH3, bodyCg);F = subs(F, LH4, bodyCg);

F = subs(F, LT1, thighCg);F = subs(F, LT2, thighCg);F = subs(F, LT3, thighCg);F = subs(F, LT4, thighCg);

F = subs(F, LS1, shankCg);F = subs(F, LS2, shankCg);TF= subs(F, LS3, shankCg);F = subs(F, LS4, shankCg);

%Moments of Inertia-------------------------------------INERTIA

syms IS1z IT1z IS2z IS3z IS4z IT2z IT3z IT4z;

T = subs(T, IS1z, iS);T = subs(T, IT1z, iT);

F = subs(F, IS1z, iS);F = subs(F, IT1z, iT);

%T = subs(T, IS2z, iS);T = subs(T, IT2z, iT);

%T = subs(T, IS3z, iS);T = subs(T, IT3z, iT);

%T = subs(T, IS4z, iS);T = subs(T, IT4z, iT);

syms th1 tk1 th1dot tk1dot th1ddot tk1ddot

Tindex = 1;

Tval = 0;

Findex = 1;

Fval = 0;

fprintf('Calculaing Average and Maximum Torque/Force...\n');

for i=1:length(ddTheta1)%Loop to get all the torques added to the sum

Ti = T;

Ti = subs(Ti, th1, theta1(i)); Ti = subs(Ti, tk1, theta2(i));

Ti = subs(Ti, th1dot, dTheta1(i)); Ti = subs(Ti, tk1dot, dTheta2(i));

Ti = subs(Ti, th1ddot, ddTheta1(i)); Ti = subs(Ti, tk1ddot, ddTheta2(i));

Fi = F;

Fi = subs(Fi, th1, theta1(i)); Fi = subs(Fi, tk1, theta2(i));

Fi = subs(Fi, th1dot, dTheta1(i)); Fi = subs(Fi, tk1dot, dTheta2(i));

Fi = subs(Fi, th1ddot, ddTheta1(i)); Fi = subs(Fi, tk1ddot, ddTheta2(i));

if i == 1

FTotal = eval(Fi);

TTotal = eval(Ti);

else

FTotal(:,i) = eval(Fi);

TTotal(:,i) = eval(Ti);

end

if mean(FTotal(:,i)) > mean(Fval) && (i > 1) %probably should find equivilant force instead of max average...

Fval = FTotal(:,i);

Findex = i;

end

if mean(TTotal(:,i)) > mean(Tval) && (i > 1)%compare the max singular torque instead?

Tval = mean(TTotal(:,i));

Tindex = i;

end

end

fprintf('Fetching robot state at critical points\n');

%Divide the torques by the number of vals - average

avgT = (trapz(t(2:length(t)-1),TTotal') / sum(dt))';

avgF = (trapz(t(2:length(t)-1),FTotal') / sum(dt))';

thetas1 = [theta1(Tindex) theta2(Tindex)];

dThetas1 = [dTheta1(Tindex) dTheta2(Tindex)];

ddThetas1 = [ddTheta1(Tindex) ddTheta2(Tindex)];

thetas2 = [theta1(Findex) theta2(Findex)];

dThetas2 = [dTheta1(Findex) dTheta2(Findex)];

ddThetas2 = [ddTheta1(Findex) ddTheta2(Findex)];

fprintf('Calculaing Forces and Torques...\n');

FI1 = F;

TI1 = T;

%Insert Values-------------------------------------------ANGLES

TI1 = subs(TI1, th1, thetas1(1)); TI1 = subs(TI1, tk1, thetas1(2));

TI1 = subs(TI1, th1dot, dThetas1(1)); TI1 = subs(TI1, tk1dot, dThetas1(2));

TI1 = subs(TI1, th1ddot, ddThetas1(1)); TI1 = subs(TI1, tk1ddot, ddThetas1(2));

%forces

FI1 = subs(FI1, th1, thetas1(1)); FI1 = subs(FI1, tk1, thetas1(2));

FI1 = subs(FI1, th1dot, dThetas1(1)); FI1 = subs(FI1, tk1dot, dThetas1(2));

FI1 = subs(FI1, th1ddot, ddThetas1(1)); FI1 = subs(FI1, tk1ddot, ddThetas1(2));

FI2 = F;

TI2 = T;

%Insert Values-------------------------------------------ANGLES

TI2 = subs(TI2, th1, thetas2(1)); TI2 = subs(TI2, tk1, thetas2(2));

TI2 = subs(TI2, th1dot, dThetas2(1)); TI2 = subs(TI2, tk1dot, dThetas2(2));

TI2 = subs(TI2, th1ddot, ddThetas2(1)); TI2 = subs(TI2, tk1ddot, ddThetas2(2));

%forces

FI2 = subs(FI2, th1, thetas2(1)); FI2 = subs(FI2, tk1, thetas2(2));

FI2 = subs(FI2, th1dot, dThetas2(1)); FI2 = subs(FI2, tk1dot, dThetas2(2));

FI2 = subs(FI2, th1ddot, ddThetas2(1)); FI2 = subs(FI2, tk1ddot, ddThetas2(2));

TI1=eval(TI1);

TI2=eval(TI2);

FI1=eval(FI1);

FI2=eval(FI2);

if ~dragging

fprintf('Calculating step impulse force\n');

vhx = bodyVel;vhy = 0; %hip velocity (x, y) from body velocity at 0 deg

%find max angular velocities

i1 = find(dTheta1==max(dTheta1)); i2 = find(dTheta2==max(dTheta2));

%calculating thigh speeds

vt1 = [thighCg\*cos(dTheta1(i1))+vhx, thighCg\*sin(dTheta1(i1))+vhy];

vt2 = [thighCg\*cos(dTheta1(i2))+vhx, thighCg\*sin(dTheta1(i2))+vhy];

%calculating knee speeds

vk1 = [L1\*cos(dTheta1(i1))+vhx, L1\*sin(dTheta1(i1))+vhy];

vk2 = [L1\*cos(dTheta1(i2))+vhx, L1\*sin(dTheta1(i2))+vhy];

%calculating shank speed

vs1 = vk1 + [L2\*cos(dTheta2(i1)), L2\*sin(dTheta2(i1))];

vs2 = vk2 + [L2\*cos(dTheta2(i2)), L2\*sin(dTheta2(i2))];

%F = mLeg \* vLeg - impulse momentum equations

Fi1 = tMass \* vt1 + sMass \* vs1;

mag1 = sqrt(Fi1(1)^2 + Fi1(2)^2);

Fi2 = tMass \* vt2 + sMass \* vs2;

mag2 = sqrt(Fi2(1)^2 + Fi2(2)^2);

fprintf('\n');

if mag1 > mag2

fprintf('Impulse force of %g N\n', mag1);

fprintf('Components: %g N x, %g N y \n', Fi1(1), Fi1(2));

else

fprintf('Impulse force of %g \n', mag2);

fprintf('Components: %g N x, %g N y \n', Fi2(1), Fi2(2));

end

end

fprintf('Simulation complete!\n');

%Function used to simulate the robotic leg

%Inputs:

%Link1, Link2 - .stl files

%L1, L2 - the length of each link

%L12 - the distance along L1 where L2 is attached

%L21 - the distance along L2 where L1 is attached

%P1x, P1y - the ground location of piston 1

%P12 - the distance of piston 1's attachment point on link 2

%p21 - the distance of piston 2's attachment point on link 1

%p22 - the distance of piston 2's attachment point on link 2

%Theta1 and Theta2 are the starting conditions

function y=LegModel(Link1, Link2, L1, L2, L12, L21, P1x, P1y, P11, P21, P22,path,sampleDistance, time)

%calculate the number of points in the path

numPts = size(path);

numPts = numPts(1);

%foot position index

index = 1;

footPosition(index,:) = path(1,:);

Theta(index,:) = InverseKinematics(L12, L2-L21, footPosition(index,1), footPosition(index,2));

drawPosition(index,:) = ForwardKinematics(L12, L2-L21, Theta(1), Theta(2));

index = index + 1;

%calculate total distance of the path

totalDistance = 0;

for i = 1:numPts

desiredPos = path(i,:);

while footPosition(index-1,1) ~= desiredPos(1) || footPosition(index-1,2) ~= desiredPos(2)

%move footPosition up to sampleDistance

d = min(sampleDistance, Distance(desiredPos(1), desiredPos(2), footPosition(index-1,1), footPosition(index-1,2)));

unitVector = [desiredPos(1) - footPosition(index-1,1),desiredPos(2) - footPosition(index-1,2)];

unitVector = unitVector ./ sqrt(unitVector(1)^2 + unitVector(2)^2);

unitVector = unitVector.\*d;

%update foot position

footPosition(index,:) = footPosition(index-1,:) + unitVector;

Theta(index,:) = InverseKinematics(L12, L2-L21, footPosition(index,1), footPosition(index,2));

drawPosition(index,:) = ForwardKinematics(L12, L2-L21, Theta(index,1), Theta(index,2));

index = index + 1;

end

if i > 1

totalDistance = totalDistance + Distance(path(i-1,1), path(i-1,2), path(i,1), path(i,2));

end

end

%calculate cylinder positions

footVelocity = totalDistance/time;

for i = 1:length(footPosition)

cylPos(i,:) = FootToCylinder(L12, L2-L21, P1x, P1y, P11, P21, P22, footPosition(i,1), footPosition(i,2));

end

for i = 1:length(footPosition)-1

d(i) = Distance(footPosition(i,1), footPosition(i,2), footPosition(i+1,1), footPosition(i+1,2));

end

dt = d / footVelocity;

assignin('base', 'dt', dt');

%calculate acceleration parameters

cylAcc(:,1) = diff(cylPos(:,1), 2);

cylAcc(:,2) = diff(cylPos(:,2), 2);

%draw the foot position

DrawArm(Link1, Link2, drawPosition(:,1), drawPosition(:,2), L12, L2-L21, cylPos, dt, cylAcc);

%

% maxTheta1 = max(Theta(:,1));

% maxTheta2 = max(Theta(:,2));

% minTheta1 = min(Theta(:,1));

% minTheta2 = min(Theta(:,2));

%

% maxPL1 = max(cylPos(:,1));

% maxPL2 = max(cylPos(:,2));

% minPL1 = min(cylPos(:,1));

% minPL2 = min(cylPos(:,2));

%

% maxP1Acc = max(abs(cylAcc(:,1)));

% maxP2Acc = max(abs(cylAcc(:,2)));

%return important values

y = Theta;%[minTheta1, maxTheta1, minTheta2, maxTheta2; ...

% minPL1, maxPL1, minPL2, maxPL2;...

% 0, maxP1Acc, 0, maxP2Acc,];

end

%Calculates the forward kinematics of a variable configuration 2R

%robotic arm

%Logan Beaver 11/28/14

%D1 = Distance along link 1 of joint 1 from the base

%D2 = Distance along link 2 of joint 1 from the foot

%T1 = Theta 1; T2 = Theta 2

function footPos = ForwardKinematics(D1,D2,T1,T2)

x = D1\*cosd(T1) + D2\*cosd(T1+T2);

y = D1\*sind(T1) + D2\*sind(T1+T2);

footPos = [x, y];

end

%The inverse kinematics to determine the robot's necessary position

%Logan Beaver ,10/4/14

function angles = InverseKinematics(L1,L2, x, y)

%"max()" is used to ensure the robot stays within its operational bounds

num = max((L1+L2)^2 - (x^2 + y^2), 0);

den = max((x^2 + y^2) - (L1-L2)^2, 0);

theta2 = -2\*atan2(sqrt(num),sqrt(den));

num = y\*(L1+L2\*cos(theta2)) - L2\*x\*sin(theta2);

den = x\*(L1+L2\*cos(theta2)) + L2\*y\*sin(theta2);

theta1 = atan2(num,den)\*180/pi;

theta2 = theta2\*180/pi;

angles = [theta1, theta2];

end

function [p,V,p2,V2] = cad2poly(filename, filename2)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Main File : cad2poly.m

% Source Files: cad2mat.m

% Description : Converts CAD geometry into multiple polygons and plots

% the resulting geometry

% Input : filename, filename2 -filenames of the geometry to convert

% Output : p, V, p2, V2 -The corresponding polygons and vertices of

% the geometry in filename and filename2, respectively.

% Author : Dr. L.A. Rodriguez

% Date : 02/04/2014

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Converts CAD data to MATLAB using the cad2ply.m file, which is a

% modified version of the cad2matdemo.m file located on the

% Mathworks central file exchange.

% F-faces, V-vertices, C-color

[F,V,C] = cad2mat(filename);

[F2,V2,C2] = cad2mat(filename2);

if strcmp(filename, 'Thigh.stl')

V(:,2) = V(:,2) - 0.445;

V(:,2) = V(:,2)\*(-1);

V(:,1) = V(:,1) - 0.05;

x = V(:,1);

V(:,1) = V(:,2);

V(:,2) = x;

V(:,1) = V(:,1);

dTheta = 33;

V = (V \* [cosd(dTheta) -sind(dTheta) 0;

sind(dTheta) cosd(dTheta) 0;

0 0 1]);

end

if strcmp(filename2, 'Shank.stl')

x2 = V2(:,1);

V2(:,1) = V2(:,2);

V2(:,2) = x2 - 0.05;

V2(:,1) = V2(:,1) \* -1 + 0.27;

end

h = figure(1);

clf;

p = patch('faces', F, 'vertices' ,V); % create the polygons

p2 = patch('faces', F2, 'vertices' ,V2); % create the polygons

set(p, 'facec', 'flat'); % Set the face color flat

set(p2, 'facec', 'flat'); % Set the face color flat

set(p, 'FaceVertexCData', C); % Set the color (from file)

set(p2, 'FaceVertexCData', C); % Set the color (from file)

set(p, 'EdgeColor','none'); % Set the edge color

set(p2, 'EdgeColor','none'); % Set the edge color

% Color options yellow,magenta,cyan,red,green,blue,white, black

set(p,'FaceColor','red') % set color of filename geometry

set(p2,'FaceColor','cyan') % set color of filename2 geometry

light % add a default light

daspect([1 1 1]) % Setting the aspect ratio

view(3) % Plot isometric view

xlabel('X'),ylabel('Y'),zlabel('Z')

%title({['Imported Solidworks Geometry from ' filename ' and ' filename2] ' Used to animate a 2-link robot'})

title(['Imported Solidworks Geometry from ' filename ' and ' filename2])

drawnow

shg

% To use homogenous transformation matrices the n by 3 vertices will be

% turned to n by 4 vertices by augmenting them with ones, [V;1]

V = [V ones(length(V),1)]';

V2 = [V2 ones(length(V2),1)]';

function [F,V,C] = cad2mat(filename)

% CAD2MATDEMO, a demonstration of importing 3D CAD data into Matlab.

% To get CAD data into Matlab, the process is:

%

% 1) Export the 3D CAD data as an ASCII STL (or Pro/E render SLP) file.

% 2) This Matlab routine reads the CAD data

% 3) Once read, the CAD data is rotated around a bit.

%

% Program has been tested with: AutoCAD, Cadkey, and Pro/Engineer.

% Should work with most any CAD programs that can export STL.

%

% Format Details: STL is supported, and the color version of STL

% that Pro/E exports, called 'render.' The render (SLP) is just

% like STL but with color added.

%

% Note: This routine has both the import function and some basic

% manipulation for testing. The actual reading mechanism is located

% at the end of this file.

if nargin == 0

filename = 'link.stl'; % a simple demo part

warning(['No file specified, using demo file: ' filename]);

end

%

% Read the CAD data file:

[F, V, C] = rndread(filename);

function [fout, vout, cout] = rndread(filename)

% Reads CAD STL ASCII files, which most CAD programs can export.

% Used to create Matlab patches of CAD 3D data.

% Returns a vertex list and face list, for Matlab patch command.

%

% filename = 'hook.stl'; % Example file.

%

fid=fopen(filename, 'r'); %Open the file, assumes STL ASCII format.

if fid == -1

error('File could not be opened, check name or path.')

end

%

% Render files take the form:

%

%solid BLOCK

% color 1.000 1.000 1.000

% facet

% normal 0.000000e+00 0.000000e+00 -1.000000e+00

% normal 0.000000e+00 0.000000e+00 -1.000000e+00

% normal 0.000000e+00 0.000000e+00 -1.000000e+00

% outer loop

% vertex 5.000000e-01 -5.000000e-01 -5.000000e-01

% vertex -5.000000e-01 -5.000000e-01 -5.000000e-01

% vertex -5.000000e-01 5.000000e-01 -5.000000e-01

% endloop

% endfacet

%

% The first line is object name, then comes multiple facet and vertex lines.

% A color specifier is next, followed by those faces of that color, until

% next color line.

%

CAD\_object\_name = sscanf(fgetl(fid), '%\*s %s'); %CAD object name, if needed.

% %Some STLs have it, some don't.

vnum=0; %Vertex number counter.

report\_num=0; %Report the status as we go.

VColor = 0;

%

while feof(fid) == 0 % test for end of file, if not then do stuff

tline = fgetl(fid); % reads a line of data from file.

fword = sscanf(tline, '%s '); % make the line a character string

% Check for color

if strncmpi(fword, 'c',1) == 1; % Checking if a "C"olor line, as "C" is 1st char.

VColor = sscanf(tline, '%\*s %f %f %f'); % & if a C, get the RGB color data of the face.

end % Keep this color, until the next color is used.

if strncmpi(fword, 'v',1) == 1; % Checking if a "V"ertex line, as "V" is 1st char.

vnum = vnum + 1; % If a V we count the # of V's

report\_num = report\_num + 1; % Report a counter, so long files show status

if report\_num > 249;

% disp(sprintf('Reading vertix num: %d.',vnum));

report\_num = 0;

end

v(:,vnum) = sscanf(tline, '%\*s %f %f %f'); % & if a V, get the XYZ data of it.

c(:,vnum) = VColor; % A color for each vertex, which will color the faces.

end % we "\*s" skip the name "color" and get the data.

end

% Build face list; The vertices are in order, so just number them.

%

fnum = vnum/3; %Number of faces, vnum is number of vertices. STL is triangles.

flist = 1:vnum; %Face list of vertices, all in order.

F = reshape(flist, 3,fnum); %Make a "3 by fnum" matrix of face list data.

%

% Return the faces and vertexs.

%

fout = F'; %Orients the array for direct use in patch.

vout = v'; % "

cout = c';

%

fclose(fid);

%calculates distance between (x1, y1) and (x2, y2)

function d = Distance(x1, y1, x2, y2)

d = sqrt((x1-x2)^2 + (y1-y2)^2);

end

function DrawArm(Link1, Link2, x, y, D1, D2, cylPos, dt, cylAcc)

% Converts CAD geometry into multiple polygons and into its vertices

% link1Pnts and link2Pnts already includes the bottom

% row of ones, that is link1Pnts = [Pnts;1]

[p,link1Pnts,p2,link2Pnts] = cad2poly(Link1, Link2);

view(2); %2D

%set plot bounds

maxVal = (max(max(max(abs(x)), max(abs(y))), 0))/100 \* 1.25;

figure(1);

hold on;

axis([-maxVal maxVal -maxVal 0.05]);

xlabel('X Position [m]');

ylabel('Y Position [m]');

cdt = cumsum(dt);

for i = 1:length(x)

Theta = InverseKinematics(D1, D2, x(i), y(i));

%update position of STL models

drawRobot(p, p2, Theta(1), Theta(2), link1Pnts, link2Pnts, (D1-2)/100);

%plot current path progress

figure(1)

movegui('northwest');

title('Robot Step Animation');

plot(x(1:i)/100, y(1:i)/100);

% pause(0.05)

%draw cylinder positions

%plot the cylinder lengths

figure(2);

movegui('north');

hold on;

subplot(2,1,1);

title('Cylinder 1 - Position vs Time [m]');

axis([0, sum(dt), min(cylPos(:,1))\*.95, max(cylPos(:,1))\*1.05]);

plot(cdt(1:min(i, length(cdt))), cylPos(1:min(i, length(dt)),1));

figure(3);

movegui('northeast');

hold on;

subplot(2,1,1);

title('Cylinder 2 - Position vs Time [m]');

axis([0, sum(dt), min(cylPos(:,2))\*.95, max(cylPos(:,2))\*1.05]);

plot(cdt(1:min(i, length(cdt))), cylPos(1:min(i, length(cdt)),2));

%plot the cylinder accelerations

figure(2);

hold on;

subplot(2,1,2);

title('Cylinder 1 - Acceleration vs Time [m/s/s]');

axis([0, sum(dt), min(cylAcc(:,1))\*.95, max(cylAcc(:,1))\*1.05]);

plot(cdt(1:min(i, length(cdt)-1)), cylAcc(1:min(i, length(cdt)-1),1));

figure(3);

hold on;

subplot(2,1,2);

title('Cylinder 2 - Acceleration vs Time [m/s/s]');

axis([0, sum(dt), min(cylAcc(:,2))\*.95, max(cylAcc(:,2))\*1.05]);

plot(cdt(1:min(i, length(cdt)-1)), cylAcc(1:min(i, length(cdt)-1),2));

end

end

function drawRobot(p,p2,theta1,theta2,link1Pnts,link2Pnts, L1)

% Rotate and translate Link vertices

T\_01 = RotZ(theta1);

link1NewPnts = T\_01\*link1Pnts; % new vertices Link1

T\_12 = Trans(L1,0,0)\*RotZ(theta2); % new vertices Link2

link2NewPnts = T\_01\*T\_12\*link2Pnts;

% Draw robot

set(p,'Vertices',link1NewPnts(1:3,:)');

set(p2,'Vertices',link2NewPnts(1:3,:)');

end

%A function that takes in the foot position and returns the length of

%cylinder 1 and cylinder 2

%Inputs

%L1 - Link 1, L2 - Link 2

%D1 - Distance along Link 1 where Link 2 is attached

%D2 - Distance along Link 2 where Link 1 is attahced

%P1x, P1y - ground position of cylinder 1

%P11 - Distance along L1 where cylidner 1 is attached

%P21 - postion of Cylinder 2 on link 1, P22 - Position of Cylinder 2 on link 2

%x, y - Foot Position

function lengths = FootToCylinder(D1, D2, P1x, P1y, P11, P21, P22, x, y)

%Calculate inverse kinematics

Theta = InverseKinematics(D1, D2, x, y);

%Length of Cylinder 1

deltaX = -P1x + P11\*cosd(90-Theta(1));

deltaY = -P1y + P11\*sind(90-Theta(1));

PL1 = Distance(deltaX, deltaY, 0, 0);

%Length of Cylinder 2

Ax = P21\*cosd(Theta(1)); %x coordinate of P21

Ay = P21\*sind(Theta(1)); %y coordinate of P21

Bx = D1\*cosd(Theta(1)) + P22\*cosd(Theta(1) + Theta(2));

By = D1\*sind(Theta(1)) + P22\*sind(Theta(1) + Theta(2));

PL2 = Distance(Ax, Ay, Bx, By);

%Store Lengths in an array

lengths = [PL1 PL2];

end

function positions = SemiEllipseStep()

halfStep = 0.25;%m

stepHeight = 0.05;%m

positions = [180:-1:0; 180:-1:0]; %initial semicircle

positions = [cos(positions(1,:)\*pi/180)\*halfStep; sin(positions(2,:)\*pi/180)\*stepHeight];

positions(2,:) = positions(2,:) - .7;

positions(1,:) = positions(1,:) + .02;

for i = 2:length(positions)-1;

positions(2,i) = (positions(2,i-1)\*2 + positions(2,i+1)\*2)/4;

end

positions = positions' \* 100;

%positions = [positions; positions(1,:)];

end

function positions = DragStep()

positions = [0.125:-0.005:-0.125; (0.125:-0.005:-0.125)\*0 - 0.55]' \* 100;

end

function data = ReadExcelData()

%IMPORTFILE Import data from a spreadsheet

% data = IMPORTFILE(FILE) reads data from the first worksheet in the

% Microsoft Excel spreadsheet file named FILE and returns the data as

% column vectors.

%

% data = IMPORTFILE(FILE,SHEET) reads from the specified worksheet.

%

% data = IMPORTFILE(FILE,SHEET,STARTROW,ENDROW) reads from the specified

% worksheet for the specified row interval(s). Specify STARTROW and

% ENDROW as a pair of scalars or vectors of matching size for

% dis-contiguous row intervals. To read to the end of the file specify an

% ENDROW of inf.%

% Example:

% data = importfile('ParameterConfig.xlsx','Sheet1',1,12);

%

% See also XLSREAD.

% Auto-generated by MATLAB on 2015/02/12 18:41:31

%% Input handling

% If no sheet is specified, read first sheet

sheet = 1;

% read 12 data points

startRow = 1;

endRow = 12;

%% Import the data

data = xlsread('ParameterConfig.xlsx', sheet, sprintf('B%d:B%d',startRow(1),endRow(1)));

for block=2:length(startRow)

tmpDataBlock = xlsread('ParameterConfig.xlsx', sheet, sprintf('B%d:B%d',startRow(block),endRow(block)));

data = [data;tmpDataBlock]; %#ok<AGROW>

end

function [ output\_args ] = MatrixDynamics( input\_args )

%DYNAMICSFUNCTION Summary of this function goes here

% Detailed explanation goes here

% \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

% / | | \

% / | | \

% / | | \

% \ \ | /

% \ \ / /

% \ \ / /

% (2) (4) (3) (1)

%

% 1 - front right, 3 - front left

% 2 - rear right, 4 - rear left

%kinematic symbols

syms x y LH1 LH2 LH3 LH4 LT1 LT2 LT3 LT4 LK1 LK2 LK3 LK4 LS1 LS2 LS3 LS4 LF1 LF2 LF3 LF4

syms tb th1 th2 th3 th4 tk1 tk2 tk3 tk4

syms xdot ydot tbdot th1dot th2dot th3dot th4dot tk1dot tk2dot tk3dot tk4dot

syms xddot yddot tbddot th1ddot th2ddot th3ddot th4ddot tk1ddot tk2ddot tk3ddot tk4ddot

%Leg 1 - FR, Leg 2 - RR, Leg 3 - FL, Leg 4 - RL

xH1 = x + LH1\*cos(2\*pi-(tb)); yH1 = y + LH1\*sin(2\*pi-(tb));

xH2 = x + LH2\*cos((tb+pi)); yH2 = y + LH2\*sin((tb+pi));

xH3 = x + LH3\*cos(2\*pi-(tb)); yH3 = y + LH3\*sin(2\*pi-(tb));

xH4 = x + LH4\*cos((tb+pi)); yH4 = y + LH4\*sin((tb+pi));

xT1 = x + LH1\*cos(2\*pi-(tb)) + LT1\*cos(2\*pi-(th1+tb)); yT1 = y + LH1\*sin(2\*pi-(tb)) + LT1\*sin(2\*pi-(th1+tb));

xT2 = x + LH2\*cos((tb+pi)) + LT2\*cos((th2+tb+pi)); yT2 = y + LH2\*sin((tb+pi)) + LT1\*sin((th2+tb+pi));

xT3 = x + LH3\*cos(2\*pi-(tb)) + LT3\*cos(2\*pi-(th3+tb)); yT3 = y + LH3\*sin(2\*pi-(tb)) + LT1\*sin(2\*pi-(th3+tb));

xT4 = x + LH4\*cos((tb+pi)) + LT4\*cos((th4+tb+pi)); yT4 = y + LH4\*sin((tb+pi)) + LT1\*sin((th4+tb+pi));

xK1 = x + LH1\*cos(2\*pi-(tb)) + LK1\*cos(2\*pi-(th1+tb)); yK1 = y + LH1\*sin(2\*pi-(tb)) + LK1\*sin(2\*pi-(th1+tb));

xK2 = x + LH2\*cos(tb+pi) + LK2\*cos(th2+tb+pi); yK2 = y + LH2\*sin(tb+pi) + LK2\*sin(th2+tb+pi);

xK3 = x + LH3\*cos(2\*pi-(tb)) + LK3\*cos(2\*pi-(th3+tb)); yK3 = y + LH3\*sin(2\*pi-(tb)) + LK3\*sin(2\*pi-(th3+tb));

xK4 = x + LH4\*cos(tb+pi) + LK4\*cos(th4+tb+pi); yK4 = y + LH4\*sin(tb+pi) + LK4\*sin(th4+tb+pi);

xS1 = x + LH1\*cos(2\*pi-(tb)) + LK1\*cos(2\*pi-(th1+tb)) + LS1\*cos(2\*pi-(tk1+th1+tb)); yS1 = y + LH1\*sin(2\*pi-(tb)) + LK1\*sin(2\*pi-(th1+tb)) + LS1\*sin(2\*pi-(tk1+th1+tb));

xS2 = x + LH2\*cos(tb+pi) + LK2\*cos(th2+tb+pi) + LS2\*cos(tk2+th2+tb+pi); yS2 = y + LH2\*sin(tb+pi) + LK2\*sin(th2+tb+pi) + LS2\*cos(tk2+th2+tb+pi);

xS3 = x + LH3\*cos(2\*pi-(tb)) + LK3\*cos(2\*pi-(th3+tb)) + LS3\*cos(2\*pi-(tk3+th3+tb)); yS3 = y + LH3\*sin(2\*pi-(tb)) + LK3\*sin(2\*pi-(th3+tb)) + LS3\*sin(2\*pi-(tk3+th3+tb));

xS4 = x + LH4\*cos(tb+pi) + LK4\*cos(th4+tb+pi) + LS4\*cos(tk4+th4+tb+pi); yS4 = y + LH4\*sin(tb+pi) + LK4\*sin(th4+tb+pi) + LS4\*cos(tk4+th4+tb+pi);

xF1 = x + LH1\*cos(2\*pi-(tb)) + LK1\*cos(2\*pi-(th1+tb)) + LF1\*cos(2\*pi-(tk1+th1+tb)); yF1 = y + LH1\*sin(2\*pi-(tb)) + LK1\*sin(2\*pi-(th1+tb)) + LF1\*sin(2\*pi-(tk1+th1+tb));

xF2 = x + LH2\*cos(tb+pi) + LK2\*cos(th2+tb+pi) + LF2\*cos(tk2+th2+tb+pi); yF2 = y + LH2\*sin(tb+pi) + LK2\*sin(th2+tb+pi) + LF2\*cos(tk2+th2+tb+pi);

xF3 = x + LH3\*cos(2\*pi-(tb)) + LK3\*cos(2\*pi-(th3+tb)) + LF3\*cos(2\*pi-(tk3+th3+tb)); yF3 = y + LH3\*sin(2\*pi-(tb)) + LK3\*sin(2\*pi-(th3+tb)) + LF3\*sin(2\*pi-(tk3+th3+tb));

xF4 = x + LH4\*cos(tb+pi) + LK4\*cos(th4+tb+pi) + LF4\*cos(tk4+th4+tb+pi); yF4 = y + LH4\*sin(tb+pi) + LK4\*sin(th4+tb+pi) + LF4\*cos(tk4+th4+tb+pi);

xH1dot = derivative(xH1, [x tb], [xdot;tbdot]);

yH1dot = derivative(yH1, [y tb], [ydot;tbdot]);

xH2dot = derivative(xH2, [x tb], [xdot;tbdot]);

yH2dot = derivative(yH2, [y tb], [ydot;tbdot]);

xH3dot = derivative(xH3, [x tb], [xdot;tbdot]);

yH3dot = derivative(yH3, [y tb], [ydot;tbdot]);

xH4dot = derivative(xH4, [x tb], [xdot;tbdot]);

yH4dot = derivative(yH4, [y tb], [ydot;tbdot]);

xT1dot = derivative(xT1, [x tb th1], [xdot;tbdot;th1dot]);

yT1dot = derivative(yT1, [y tb th1], [ydot;tbdot;th1dot]);

xT2dot = derivative(xT2, [x tb th2], [xdot;tbdot;th2dot]);

yT2dot = derivative(yT2, [y tb th2], [ydot;tbdot;th2dot]);

xT3dot = derivative(xT3, [x tb th3], [xdot;tbdot;th3dot]);

yT3dot = derivative(yT3, [y tb th3], [ydot;tbdot;th3dot]);

xT4dot = derivative(xT4, [x tb th4], [xdot;tbdot;th4dot]);

yT4dot = derivative(yT4, [y tb th4], [ydot;tbdot;th4dot]);

xK1dot = derivative(xK1, [x tb th1], [xdot;tbdot;th1dot]);

yK1dot = derivative(yK1, [y tb th1], [ydot;tbdot;th1dot]);

xK2dot = derivative(xK2, [x tb th2], [xdot;tbdot;th2dot]);

yK2dot = derivative(yK2, [y tb th2], [ydot;tbdot;th2dot]);

xK3dot = derivative(xK3, [x tb th3], [xdot;tbdot;th3dot]);

yK3dot = derivative(yK3, [y tb th3], [ydot;tbdot;th3dot]);

xK4dot = derivative(xK4, [x tb th4], [xdot;tbdot;th4dot]);

yK4dot = derivative(yK4, [y tb th4], [ydot;tbdot;th4dot]);

xS1dot = derivative(xS1, [x tb th1 tk1], [xdot;tbdot;th1dot;tk1dot]);

yS1dot = derivative(yS1, [y tb th1 tk1], [ydot;tbdot;th1dot;tk1dot]);

xS2dot = derivative(xS2, [x tb th2 tk2], [xdot;tbdot;th2dot;tk2dot]);

yS2dot = derivative(yS2, [y tb th2 tk2], [ydot;tbdot;th2dot;tk2dot]);

xS3dot = derivative(xS3, [x tb th3 tk3], [xdot;tbdot;th3dot;tk3dot]);

yS3dot = derivative(yS3, [y tb th3 tk3], [ydot;tbdot;th3dot;tk3dot]);

xS4dot = derivative(xS4, [x tb th4 tk4], [xdot;tbdot;th4dot;tk4dot]);

yS4dot = derivative(yS4, [y tb th4 tk4], [ydot;tbdot;th4dot;tk4dot]);

xF1dot = derivative(xF1, [x tb th1 tk1], [xdot;tbdot;th1dot;tk1dot]);

yF1dot = derivative(yF1, [y tb th1 tk1], [ydot;tbdot;th1dot;tk1dot]);

xF2dot = derivative(xF2, [x tb th2 tk2], [xdot;tbdot;th2dot;tk2dot]);

yF2dot = derivative(yF2, [y tb th2 tk2], [ydot;tbdot;th2dot;tk2dot]);

xF3dot = derivative(xF3, [x tb th3 tk3], [xdot;tbdot;th3dot;tk3dot]);

yF3dot = derivative(yF3, [y tb th3 tk3], [ydot;tbdot;th3dot;tk3dot]);

xF4dot = derivative(xF4, [x tb th4 tk4], [xdot;tbdot;th4dot;tk4dot]);

yF4dot = derivative(yF4, [y tb th4 tk4], [ydot;tbdot;th4dot;tk4dot]);

%calculating accelerations

%hip

xH1ddot = derivative(xH1dot,[x tb xdot tbdot], [xdot;tbdot;xddot;tbddot]); %Hip 1 X

yH1ddot = derivative(yH1dot,[y tb ydot tbdot], [ydot;tbdot;yddot;tbddot]); %Hip 1 Y

xH2ddot = derivative(xH2dot,[x tb xdot tbdot], [xdot;tbdot;xddot;tbddot]); %Hip 2 X

yH2ddot = derivative(yH2dot,[y tb ydot tbdot], [ydot;tbdot;yddot;tbddot]); %Hip 2 Y

xH3ddot = derivative(xH3dot,[x tb xdot tbdot], [xdot;tbdot;xddot;tbddot]); %Hip 3 X

yH3ddot = derivative(yH3dot,[y tb ydot tbdot], [ydot;tbdot;yddot;tbddot]); %Hip 3 Y

xH4ddot = derivative(xH4dot,[x tb xdot tbdot], [xdot;tbdot;xddot;tbddot]); %Hip 4 X

yH4ddot = derivative(yH4dot,[y tb ydot tbdot], [ydot;tbdot;yddot;tbddot]); %Hip 4 Y

%thigh

xT1ddot = derivative(xT1dot, [x tb th1 xdot tbdot th1dot], [xdot;tbdot;th1dot;xddot;tbddot;th1ddot]); %Thigh 1 X

yT1ddot = derivative(yT1dot, [y tb th1 ydot tbdot th1dot], [ydot;tbdot;th1dot;yddot;tbddot;th1ddot]); %Thigh 1 Y

xT2ddot = derivative(xT2dot, [x tb th2 xdot tbdot th2dot], [xdot;tbdot;th2dot;xddot;tbddot;th2ddot]); %Thigh 2 X

yT2ddot = derivative(yT2dot, [y tb th2 ydot tbdot th2dot], [ydot;tbdot;th2dot;yddot;tbddot;th2ddot]); %Thigh 2 Y

xT3ddot = derivative(xT3dot, [x tb th3 xdot tbdot th3dot], [xdot;tbdot;th3dot;xddot;tbddot;th3ddot]); %Thigh 3 X

yT3ddot = derivative(yT3dot, [y tb th3 ydot tbdot th3dot], [ydot;tbdot;th3dot;yddot;tbddot;th3ddot]); %Thigh 3 Y

xT4ddot = derivative(xT4dot, [x tb th4 xdot tbdot th4dot], [xdot;tbdot;th4dot;xddot;tbddot;th4ddot]); %Thigh 4 X

yT4ddot = derivative(yT4dot, [y tb th4 ydot tbdot th4dot], [ydot;tbdot;th4dot;yddot;tbddot;th4ddot]); %Thigh 4 Y

%knee

xK1ddot = derivative(xK1dot,[x tb th1 xdot tbdot th1dot], [xdot;tbdot;th1dot;xddot;tbddot;th1ddot]); %Knee 1 X

yK1ddot = derivative(yK1dot,[y tb th1 ydot tbdot th1dot], [ydot;tbdot;th1dot;yddot;tbddot;th1ddot]); %Knee 1 Y

xK2ddot = derivative(xK2dot,[x tb th2 xdot tbdot th2dot], [xdot;tbdot;th2dot;xddot;tbddot;th2ddot]); %Knee 2 X

yK2ddot = derivative(yK2dot,[y tb th2 ydot tbdot th2dot], [ydot;tbdot;th2dot;yddot;tbddot;th2ddot]); %Knee 2 Y

xK3ddot = derivative(xK3dot,[x tb th3 xdot tbdot th3dot], [xdot;tbdot;th3dot;xddot;tbddot;th3ddot]); %Knee 3 X

yK3ddot = derivative(yK3dot,[y tb th3 ydot tbdot th3dot], [ydot;tbdot;th3dot;yddot;tbddot;th3ddot]); %Knee 3 Y

xK4ddot = derivative(xK4dot,[x tb th4 xdot tbdot th4dot], [xdot;tbdot;th4dot;xddot;tbddot;th4ddot]); %Knee 4 X

yK4ddot = derivative(yK4dot,[y tb th4 ydot tbdot th4dot], [ydot;tbdot;th4dot;yddot;tbddot;th4ddot]); %Knee 4 Y

%shank

xS1ddot = derivative(xS1dot,[x tb th1 tk1 xdot tbdot th1dot tk1dot], [xdot;tbdot;th1dot;tk1dot;xddot;tbddot;th1ddot;tk1ddot]); %Shank 1 X

yS1ddot = derivative(yS1dot,[y tb th1 tk1 ydot tbdot th1dot tk1dot], [ydot;tbdot;th1dot;tk1dot;yddot;tbddot;th1ddot;tk1ddot]); %Shank 1 Y

xS2ddot = derivative(xS2dot,[x tb th2 tk2 xdot tbdot th2dot tk2dot], [xdot;tbdot;th2dot;tk2dot;xddot;tbddot;th2ddot;tk2ddot]); %Shank 2 X

yS2ddot = derivative(yS2dot,[y tb th2 tk2 ydot tbdot th2dot tk2dot], [ydot;tbdot;th2dot;tk2dot;yddot;tbddot;th2ddot;tk2ddot]); %Shank 2 Y

xS3ddot = derivative(xS3dot,[x tb th3 tk3 xdot tbdot th3dot tk3dot], [xdot;tbdot;th3dot;tk3dot;xddot;tbddot;th3ddot;tk3ddot]); %Shank 3 X

yS3ddot = derivative(yS3dot,[y tb th3 tk3 ydot tbdot th3dot tk3dot], [ydot;tbdot;th3dot;tk3dot;yddot;tbddot;th3ddot;tk3ddot]); %Shank 3 Y

xS4ddot = derivative(xS4dot,[x tb th4 tk4 xdot tbdot th4dot tk4dot], [xdot;tbdot;th4dot;tk4dot;xddot;tbddot;th4ddot;tk4ddot]); %Shank 4 X

yS4ddot = derivative(yS4dot,[y tb th4 tk4 ydot tbdot th4dot tk4dot], [ydot;tbdot;th4dot;tk4dot;yddot;tbddot;th4ddot;tk4ddot]); %Shank 4 Y

%foot

xF1ddot = derivative(xF1dot,[x tb th1 tk1 xdot tbdot th1dot tk1dot], [xdot;tbdot;th1dot;tk1dot;xddot;tbddot;th1ddot;tk1ddot]); %Foot 1 X

yF1ddot = derivative(yF1dot,[y tb th1 tk1 ydot tbdot th1dot tk1dot], [ydot;tbdot;th1dot;tk1dot;yddot;tbddot;th1ddot;tk1ddot]); %Foot 1 Y

xF2ddot = derivative(xF2dot,[x tb th2 tk2 xdot tbdot th2dot tk2dot], [xdot;tbdot;th2dot;tk2dot;xddot;tbddot;th2ddot;tk2ddot]); %Foot 2 X

yF2ddot = derivative(yF2dot,[y tb th2 tk2 ydot tbdot th2dot tk2dot], [ydot;tbdot;th2dot;tk2dot;yddot;tbddot;th2ddot;tk2ddot]); %Foot 2 Y

xF3ddot = derivative(xF3dot,[x tb th3 tk3 xdot tbdot th3dot tk3dot], [xdot;tbdot;th3dot;tk3dot;xddot;tbddot;th3ddot;tk3ddot]); %Foot 3 X

yF3ddot = derivative(yF3dot,[y tb th3 tk3 ydot tbdot th3dot tk3dot], [ydot;tbdot;th3dot;tk3dot;yddot;tbddot;th3ddot;tk3ddot]); %Foot 3 Y

xF4ddot = derivative(xF4dot,[x tb th4 tk4 xdot tbdot th4dot tk4dot], [xdot;tbdot;th4dot;tk4dot;xddot;tbddot;th4ddot;tk4ddot]); %Foot 4 X

yF4ddot = derivative(yF4dot,[y tb th4 tk4 ydot tbdot th4dot tk4dot], [ydot;tbdot;th4dot;tk4dot;yddot;tbddot;th4ddot;tk4ddot]); %Foot 4 Y

%Dynamic calculations

syms mB mT1 mT2 mT3 mT4 mS1 mS2 mS3 mS4 g; %mass and weight

%Moments of inertia

syms IS1x IS1y IS1z IT1x IT1y IT1z IBx IBy IBz IT2x IT2y IT2z IS2x IS2y IS2z; %Link moments of inertia

syms IS3x IS3y IS3z IT3x IT3y IT3z IT4x IT4y IT4z IS4x IS4y IS4z; %Link moments of inertia

%Forces

syms FH1x FH2x FH3x FH4x FK1x FK2x FK3x FK4x FF1x FF2x FF3x FF4x;

syms FH1y FH2y FH3y FH4y FK1y FK2y FK3y FK4y FF1y FF2y FF3y FF4y;

%torques

syms TH1 TH2 TH3 TH4 TK1 TK2 TK3 TK4;

%weight equations

T1w = [0 -mT1 \* g 0]; S1w = [0 -mS1 \* g 0]; %weights

T2w = [0 -mT2 \* g 0]; S2w = [0 -mS2 \* g 0]; %weights

T3w = [0 -mT3 \* g 0]; S3w = [0 -mS3 \* g 0]; %weights

T4w = [0 -mT4 \* g 0]; S4w = [0 -mS4 \* g 0]; %weights

%force vectors

FH1 = [FH1x FH1y 0]; FH2 = [FH2x FH2y 0]; FH3 = [FH3x FH3y 0]; FH4 = [FH4x FH4y 0];

FK1 = [FK1x FK1y 0]; FK2 = [FK2x FK2y 0]; FK3 = [FK3x FK3y 0]; FK4 = [FK4x FK4y 0];

FF1 = [FF1x FF1y 0]; FF2 = [FF2x FF2y 0]; FF3 = [FF3x FF3y 0]; FF4 = [FF4x FF4y 0];

%vector positions and accelerations for each point of interest

%front (1)

rH1 = [xH1 yH1 0];% rH1ddot = [xH1ddot yH1ddot 0]; %hip 1

rT1 = [xT1 yT1 0]; rT1ddot = [xT1ddot yT1ddot 0]; %thigh 1

rK1 = [xK1 yK1 0];% rK1ddot = [xK1ddot yK1ddot 0]; %knee 1

rS1 = [xS1 yS1 0]; rS1ddot = [xS1ddot yS1ddot 0]; %shank 1

rF1 = [xF1 yF1 0];% rF1ddot = [xF1ddot yF1ddot 0]; %foot 1

%front (3)

rH3 = [xH3 yH3 0];% rH3ddot = [xH3ddot yH3ddot 0]; %hip 3

rT3 = [xT3 yT3 0]; rT3ddot = [xT3ddot yT3ddot 0]; %thigh 3

rK3 = [xK3 yK3 0];% rK3ddot = [xK3ddot yK3ddot 0]; %knee 3

rS3 = [xS3 yS3 0]; rS3ddot = [xS3ddot yS3ddot 0]; %shank 3

rF3 = [xF3 yF3 0];% rF3ddot = [xF3ddot yF3ddot 0]; %foot 3

%body

rB = [x y 0]; rBddot = [xddot yddot 0]; %body

%rear (2)

rH2 = [xH2 yH2 0];% rH2ddot = [xH2ddot yH2ddot 0]; %hip 2

rT2 = [xT2 yT2 0]; rT2ddot = [xT2ddot yT2ddot 0]; %thigh 2

rK2 = [xK2 yK2 0];% rK2ddot = [xK2ddot yK2ddot 0]; %knee 2

rS2 = [xS2 yS2 0]; rS2ddot = [xS2ddot yS2ddot 0]; %shank 2

rF2 = [xF2 yF2 0];% rF2ddot = [xF2ddot yF2ddot 0]; %foot 2

%rear (4)

rH4 = [xH4 yH4 0];% rH4ddot = [xH4ddot yH4ddot 0]; %hip 4

rT4 = [xT4 yT4 0]; rT4ddot = [xT4ddot yT4ddot 0]; %thigh 4

rK4 = [xK4 yK4 0];% rK4ddot = [xK4ddot yK4ddot 0]; %knee 4

rS4 = [xS4 yS4 0]; rS4ddot = [xS4ddot yS4ddot 0]; %shank 4

rF4 = [xF4 yF4 0];% rF4ddot = [xF4ddot yF4ddot 0]; %foot 4

%force calculations

EffecMomentS1 = cross(rS1,mS1\*rS1ddot)+ [IS1x\*0 IS1y\*0 IS1z\*tk1ddot]; %shank 1

EffecMomentT1 = cross(rT1,mT1\*rT1ddot)+ [IT1x\*0 IT1y\*0 IT1z\*th1ddot]; %thigh 1

EffecMomentS3 = cross(rS3,mS3\*rS3ddot)+ [IS3x\*0 IS3y\*0 IS3z\*tk3ddot]; %shank 3

EffecMomentT3 = cross(rT3,mT3\*rT3ddot)+ [IT3x\*0 IT3y\*0 IT3z\*th3ddot]; %thigh 3

%EffecMomentB = cross(rB,mB\*rBddot) + [IBx\*0 IBy\*0 IBz\*tbddot]; %body

EffecMomentT2 = cross(rT2,mT2\*rT2ddot)+ [IT2x\*0 IT2y\*0 IT2z\*th2ddot]; %thigh 2

EffecMomentS2 = cross(rS2,mS2\*rS2ddot)+ [IS2x\*0 IS2y\*0 IS2z\*tk2ddot]; %shank 2

EffecMomentT4 = cross(rT4,mT4\*rT4ddot)+ [IT4x\*0 IT4y\*0 IT4z\*th4ddot]; %thigh 2

EffecMomentS4 = cross(rS4,mS4\*rS4ddot)+ [IS4x\*0 IS4y\*0 IS4z\*tk4ddot]; %shank 2

%creating the 1D Force Matrix {F}

F = [FK1y; FH1y; FK2y; FH2y; FK3y; FH3y; FK4y; FH4y; ...

FK1x; FH1x; FK2x; FH2x; FK3x; FH3x; FK4x; FH4x];

%The "other crap" on the right half {B}

Kin = [mS1\*yS1ddot+S1w(2)+FF1y; mT1\*yT1ddot+T1w(2); mS2\*yS2ddot+S2w(2)+FF2y; mT2\*yT2ddot+T2w(2); ...

mS3\*yS3ddot+S3w(2)+FF3y; mT3\*yT3ddot+T3w(2); mS4\*yS4ddot+S4w(2)+FF4y; mT4\*yT4ddot+T4w(2); ...

mS1\*xS1ddot+FF1x; mT1\*xT1ddot; mS2\*xS2ddot+FF2x; mT2\*xT2ddot; ...

mS3\*xS3ddot+FF3x; mT3\*xT3ddot; mS4\*xS4ddot+FF4x; mT4\*xT4ddot];

%The matrix relating internal forces to accelerations {A}

Mat =[-1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;

1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0;

0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0;

0 0 1 -1 0 0 0 0 0 0 0 0 0 0 0 0;

0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0;

0 0 0 0 1 -1 0 0 0 0 0 0 0 0 0 0;

0 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0;

0 0 0 0 0 0 1 -1 0 0 0 0 0 0 0 0;

0 0 0 0 0 0 0 0 -1 0 0 0 0 0 0 0;

0 0 0 0 0 0 0 0 1 -1 0 0 0 0 0 0;

0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 0;

0 0 0 0 0 0 0 0 0 0 1 -1 0 0 0 0;

0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0;

0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0 0;

0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0;

0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1];

%solving {A}{F}={B} for {F}

FORCES = solve(F == inv(Mat)\*Kin, FK1x, FK1y, FK2x, FK2y, FK3x, FK3y, FK4x, FK4y, ...

FH1x, FH1y, FH2x, FH2y, FH3x, FH3y, FH4x, FH4y);

%Torque array {T}

%These are negative because on the diagram the moments are assumed negative

T = [-TK1; -TH1; -TK2; -TH2; -TK3; -TH3; -TK4; -TH4];

%Matric {C} relating torques to movement

TMat = [1 0 0 0 0 0 0 0;

-1 1 0 0 0 0 0 0;

0 0 1 0 0 0 0 0;

0 0 -1 1 0 0 0 0;

0 0 0 0 1 0 0 0;

0 0 0 0 -1 1 0 0;

0 0 0 0 0 0 1 0;

0 0 0 0 0 0 -1 1];

%The "other crap" from the sum of the torques, matrix {D}

TorqueT = [EffecMomentS1 - cross(rF1, FF1) - cross(rK1, -FK1) - cross(rS1, S1w);

EffecMomentT1 - cross(rH1, -FH1) - cross(rK1, FK1) - cross(rT1, T1w);

EffecMomentS2 - cross(rF2, FF2) - cross(rK2, -FK2) - cross(rS2, S2w);

EffecMomentT2 - cross(rH2, -FH2) - cross(rK2, FK2) - cross(rT2, T2w);

EffecMomentS3 - cross(rF3, FF3) - cross(rK3, -FK3) - cross(rS3, S3w);

EffecMomentT3 - cross(rH3, -FH3) - cross(rK3, FK3) - cross(rT3, T3w);

EffecMomentS4 - cross(rF4, FF4) - cross(rK4, -FK4) - cross(rS4, S4w);

EffecMomentT4 - cross(rH4, -FH4) - cross(rK4, FK4) - cross(rT4, T4w)];

%Grabbing the Z direction of {D}

TorqueT = TorqueT(:,3);

%Solving {C}{T}={D} for {T}

Ts = solve(TMat\*T == TorqueT, TK1, TH1, TK2, TH2, TK3, TH3, TK4, TH4);

%Creating TH1F (Hip Torque 1) in the form shown by Dr Rodriguez by renaming and zeroing

%variables

TH1F = Ts.TH1;

TH1F = subs(TH1F, ydot, 0); TH1F = subs(TH1F, yddot, 0); TH1F = subs(TH1F, th1dot, 0);

TH1F = subs(TH1F, th1ddot, 0); TH1F = subs(TH1F, tk1dot, 0); TH1F = subs(TH1F, tk1ddot, 0);

TH1F = subs(TH1F, tbdot, 0); TH1F = subs(TH1F, tbddot, 0);

TH1F = subs(TH1F, xdot, 0); TH1F = subs(TH1F, xddot, 0);

TH1F = subs(TH1F, LH1, 0);

TH1F = subs(TH1F, tb, 0);

TH1F = subs(TH1F, x, 0);

TH1F = subs(TH1F, y, 0);

TH1F = subs(TH1F, LS1, LF1/2);

TH1F = subs(TH1F, LT1, LK1/2);

%Creating TK1F (Knee Torque 1) in the form shown by Dr Rodriguez by renaming and zeroing

%variables

TK1F = Ts.TK1;

TK1F = subs(TK1F, ydot, 0); TK1F = subs(TK1F, yddot, 0); TK1F = subs(TK1F, th1dot, 0);

TK1F = subs(TK1F, th1ddot, 0); TK1F = subs(TK1F, tk1dot, 0); TK1F = subs(TK1F, tk1ddot, 0);

TK1F = subs(TK1F, tbdot, 0); TK1F = subs(TK1F, tbddot, 0);

TK1F = subs(TK1F, xdot, 0); TK1F = subs(TK1F, xddot, 0);

TK1F = subs(TK1F, LH1, 0);

TK1F = subs(TK1F, tb, 0);

TK1F = subs(TK1F, x, 0);

TK1F = subs(TK1F, y, 0);

%Creating TH2F (Hip Torque 2) in the form shown by Dr Rodriguez by renaming and zeroing

%variables

TH2F = Ts.TH2;

TH2F = subs(TH2F, ydot, 0); TH2F = subs(TH2F, yddot, 0); TH2F = subs(TH2F, th2dot, 0);

TH2F = subs(TH2F, th2ddot, 0); TH2F = subs(TH2F, tk2dot, 0); TH2F = subs(TH2F, tk2ddot, 0);

TH2F = subs(TH2F, tbdot, 0); TH2F = subs(TH2F, tbddot, 0);

TH2F = subs(TH2F, xdot, 0); TH2F = subs(TH2F, xddot, 0);

TH2F = subs(TH2F, LH2, 0);

TH2F = subs(TH2F, tb, 0);

TH2F = subs(TH2F, x, 0);

TH2F = subs(TH2F, y, 0);

%Creating TK1F (Knee Torque 1) in the form shown by Dr Rodriguez by renaming and zeroing

%variables

TK2F = Ts.TK2;

TK2F = subs(TK2F, ydot, 0); TK2F = subs(TK2F, yddot, 0); TK2F = subs(TK2F, th2dot, 0);

TK2F = subs(TK2F, th2ddot, 0); TK2F = subs(TK2F, tk2dot, 0); TK2F = subs(TK2F, tk2ddot, 0);

TK2F = subs(TK2F, tbdot, 0); TK2F = subs(TK2F, tbddot, 0);

TK2F = subs(TK2F, xdot, 0); TK2F = subs(TK2F, xddot, 0);

TK2F = subs(TK2F, LH2, 0);

TK2F = subs(TK2F, tb, 0);

TK2F = subs(TK2F, x, 0);

TK2F = subs(TK2F, y, 0);

%Print form variables to console

TH1F;

TK1F;

TH2F;

TK2F;

%Assigning variables to workspace

assignin('base', 'TK1F', TK1F);

assignin('base', 'TH1F', TH1F);

%Assigning force matrices to workspace

assignin('base', 'Kin', Kin);

assignin('base', 'F', F);

assignin('base', 'Mat', Mat);

assignin('base', 'FORCES', FORCES);

%Assigning torque matrices to workspace

assignin('base', 'T', T);

assignin('base', 'TorqueT', TorqueT);

assignin('base', 'TMat', TMat);

assignin('base', 'Ts', Ts);

end

%inputs the function eqtn, row array of variables, column array of variable

%time derivatives

function dt = derivative(eqtn, variables, timeDerivs)

%calculates the row-major jacobian of the equation and multiplies it by the

%time derivative of each varial to find the equations derivative

dt = jacobian(eqtn,variables)\*timeDerivs;

end