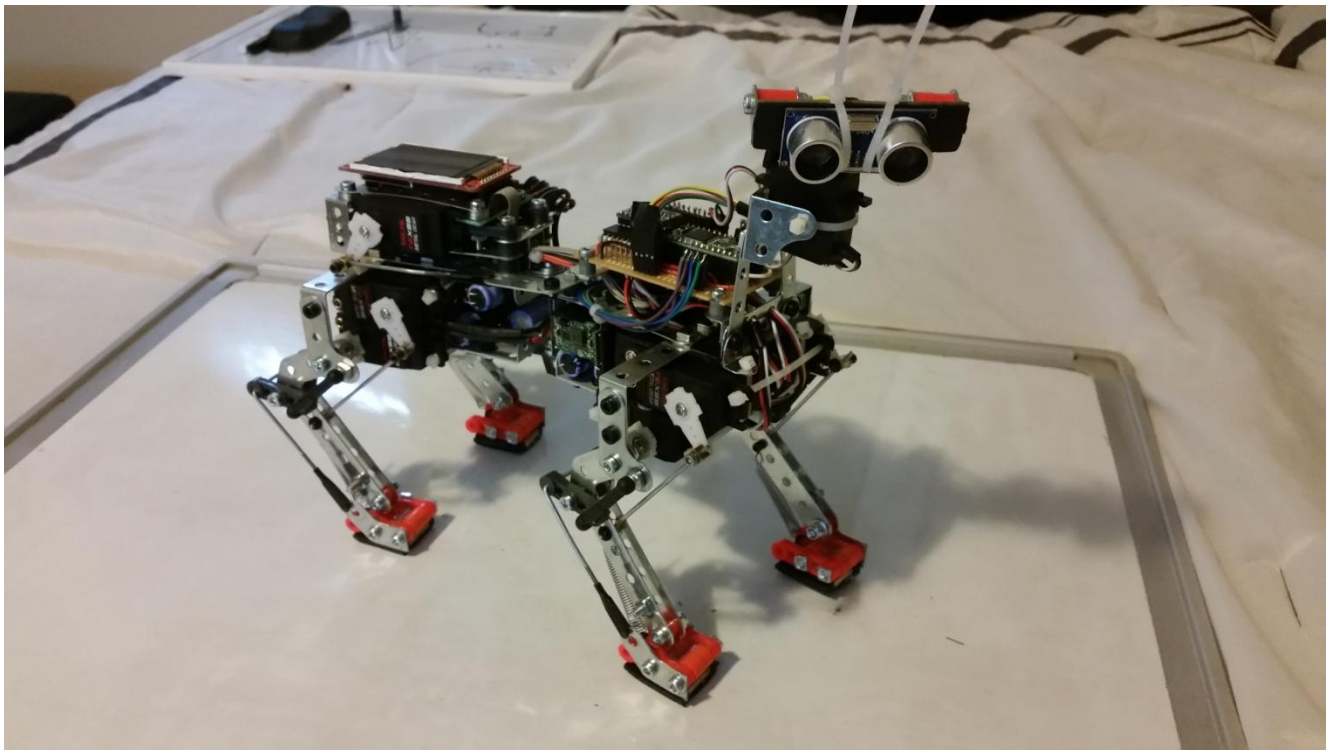
Robot Dog

January 2016

Alex Vassallo



1 Abstract

This design was created as a midterm project for the CSE466 Embedded Systems class at the University of Washington. The assignment was to implement any design that utilized the Teensy 3.1 microcontroller and various peripherals of our choice, within 2 weeks. I decided to use my mechanical skills to develop a robot that uses PWM servos. For mobility, the overall design of this project was based on my dog. Each leg is connected to two heavy-duty servos which allows precise control of each foot placement. The power grid was designed to deliver up to 25 Amps of current to manipulate each of the leg servos to do various commands. There is an additional servo mounted to the pivot point at the waist that allows up to 30 degrees of left/right rotation, and another servo mounted to a "neck" to rotate a range finder and produce a radar map to locate open routes and objects. The accelerometer and gyro unit, located on the system board, sends data to a quaternion algorithm to calculate yaw, pitch, and roll. This allows for a feedback system to keep the robot level by adjusting the height of each leg. An LCD screen is used to display data, including line drawings of each leg's current position. Currently, commands are entered into a startup script inside the source code, but development of a wireless command system is in progress.

2 Hardware

2.1 System Schematic

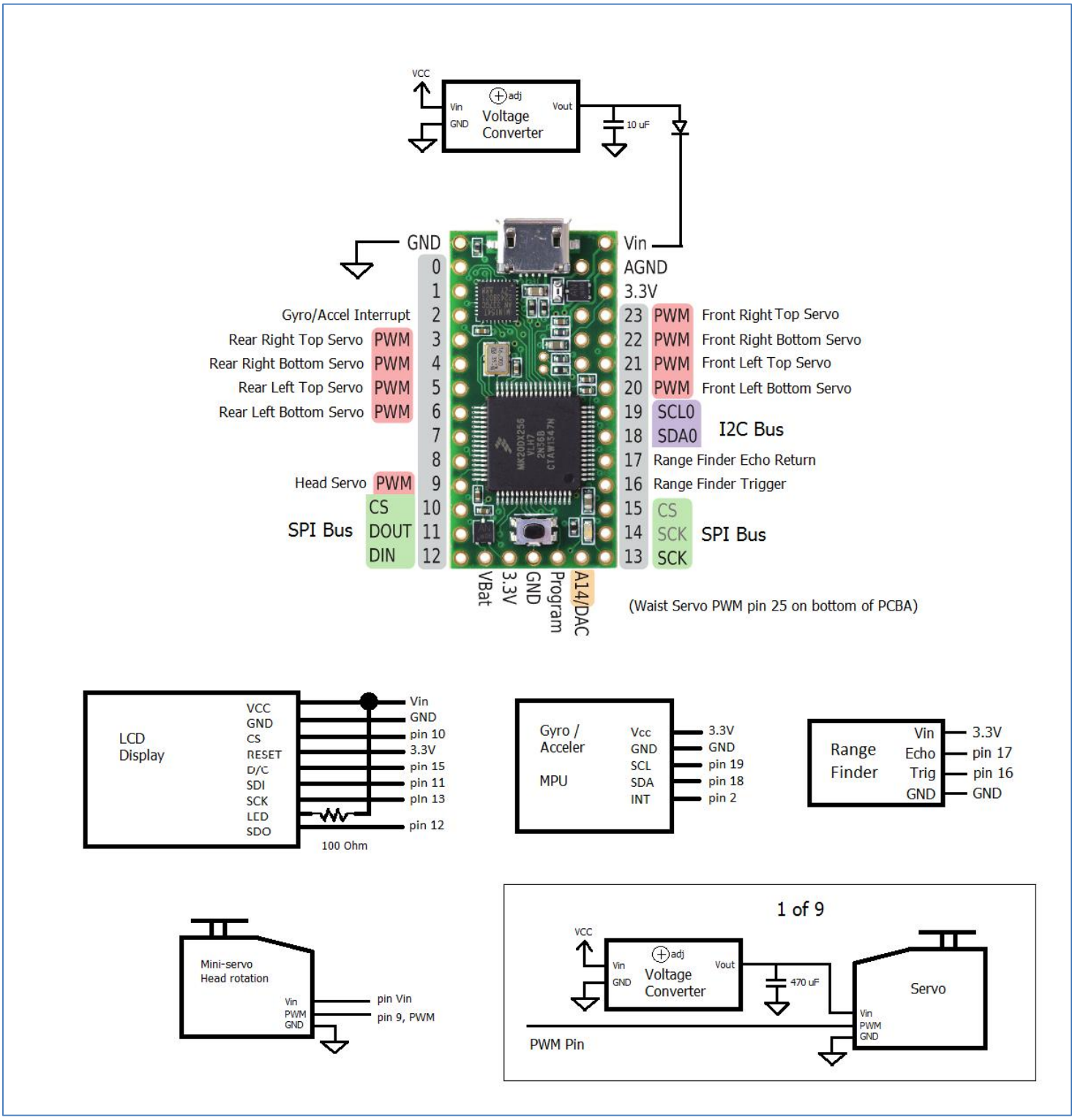


Figure 1: Schematic

2.2 Frame

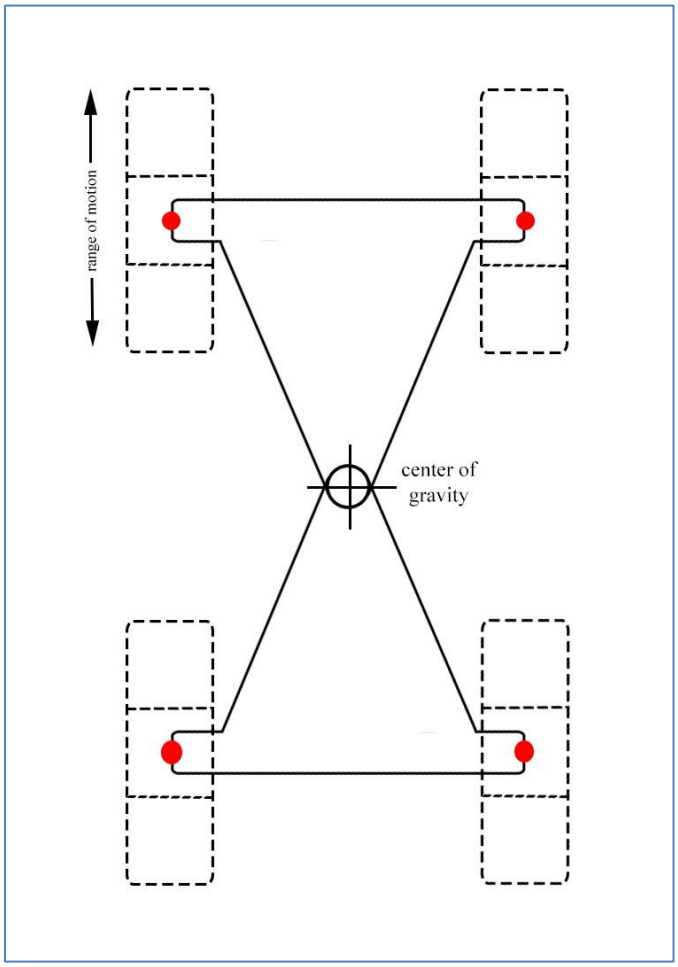
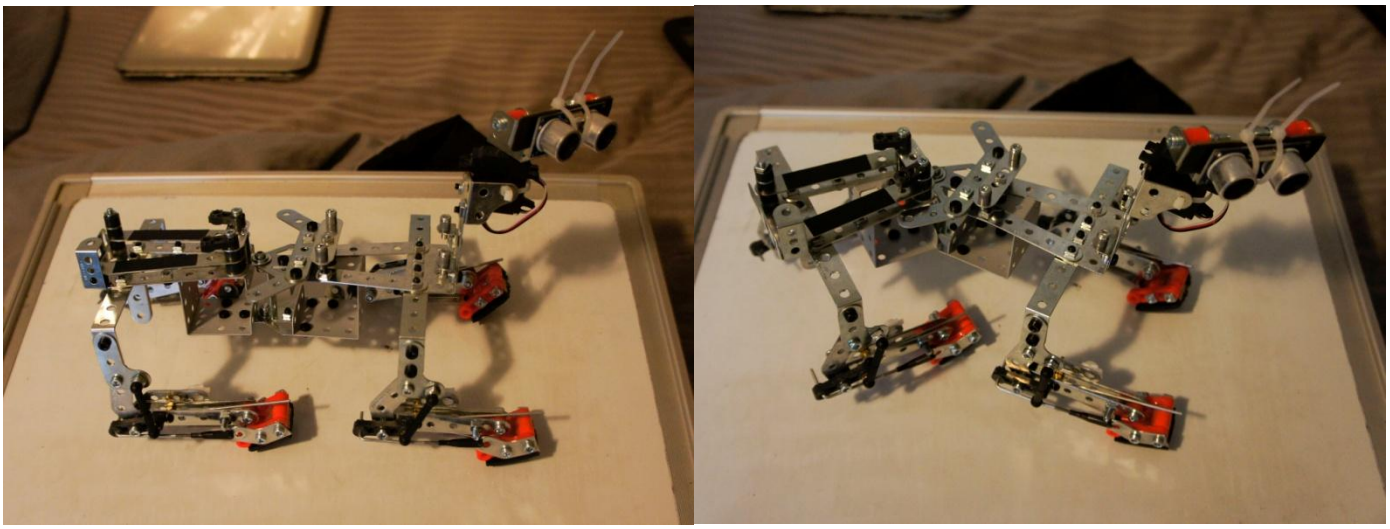


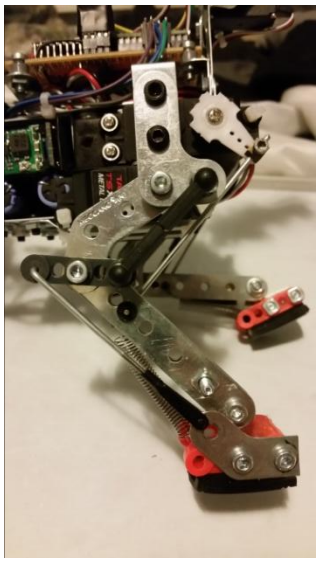
Figure 2: Frame

The prototype frame was constructed from modified Erector set parts and various linkages found at a local hobby store. It was designed to maintain a center of gravity that was low and in the center of the leg distribution. Figure 2 shows the geometry of the frame, and the red dots represent the feet inside of their range of motion with contact along the ground. In this diagram, the feet are placed in the center of motion and the center of gravity is marked at the pivot point of the frame, which is also geometrically the center of the foot distribution. By pivoting the frame, the center of gravity can be altered left or right. By placing all the hardware evenly along the frame, the weight distribution can be evenly maintained and gyroscopic balance can be used to achieve complex motions such as walking.



Empty Metal Frame

2.3 Legs



The hardware for the legs is modeled from my pet dog and includes various linkages to reduce the amount of muscle inputs needed. The target of this design was to allow for a wide range of motion while keeping the feet in alignment with the ground. Pivot points were carefully calculated to allow the servos full range of motion as they manipulate the legs. Figure 3 shows some various positions and how the hardware functions together. The servo arms are in Green, hard linkages are in Red, and Blue dots represent load bearing bushings.

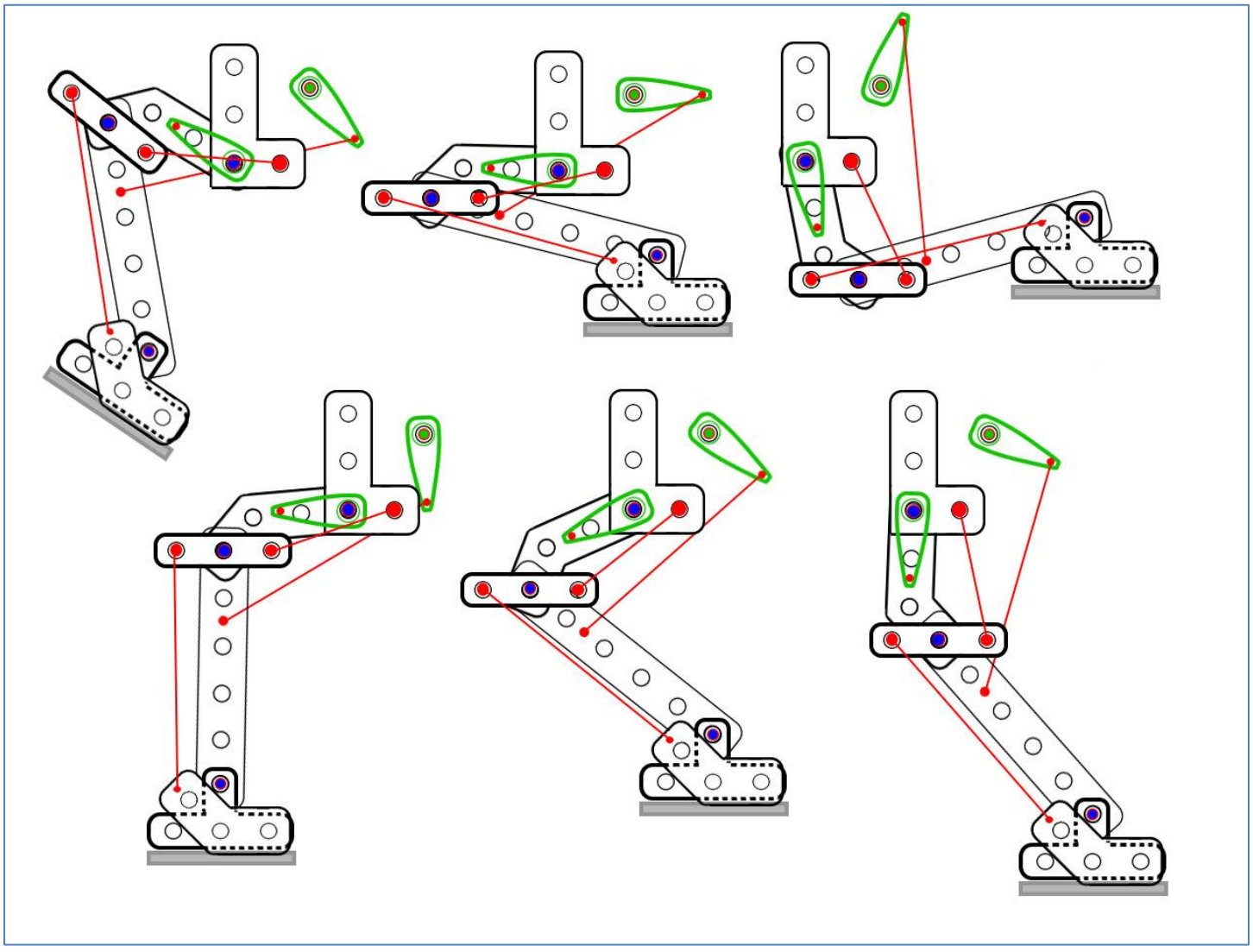
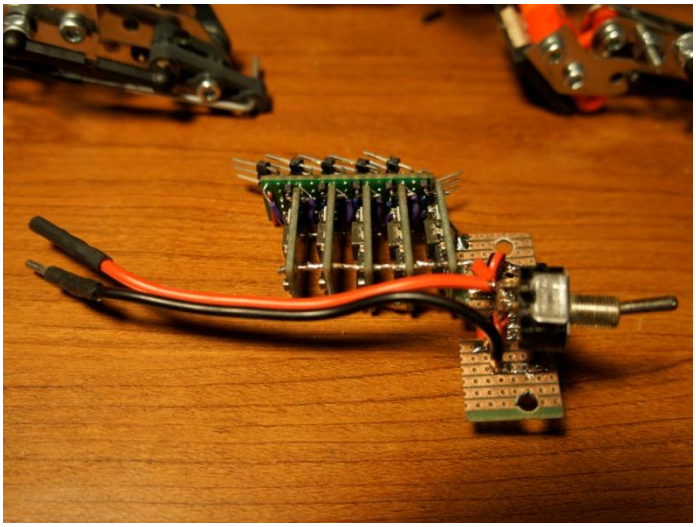


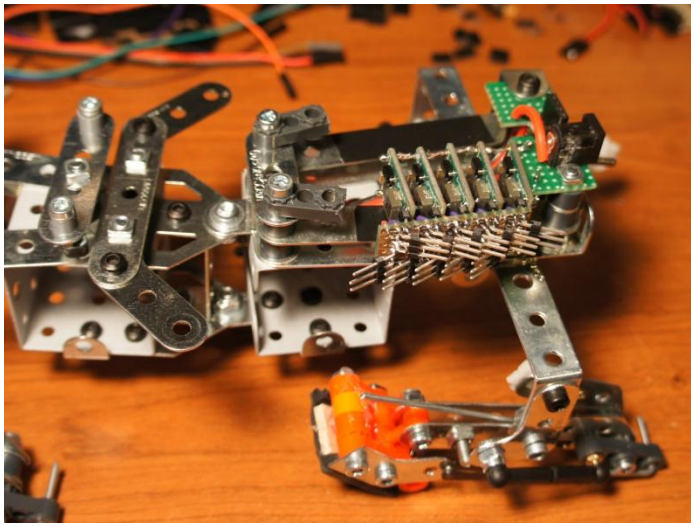
Figure 3: Leg Motion

2.4 Rear Power Grid

The servo specifications show that their supply voltage can range from 4.2V to 6V and the maximum current draw is 3 Amps each. To provide this power to each servo, adjustable voltage regulation in the form of step-down converters are connected to each servo's power supply. The output voltage of each voltage converter is adjusted to 5.5V to allow maximum torque at each servo, while leaving 0.5V of room for inductive spikes from the motors. Step-down converters were chosen for their efficiency to reduce overhead power loss as well as heat production. Each voltage converter is soldered together into a "stack", and then a junction board with external connections is soldered onto one edge. To allow the power switch and power plug to be accessible from the rear, they were attached directly to the rear PCB assembly to allow solid connections to the power source.

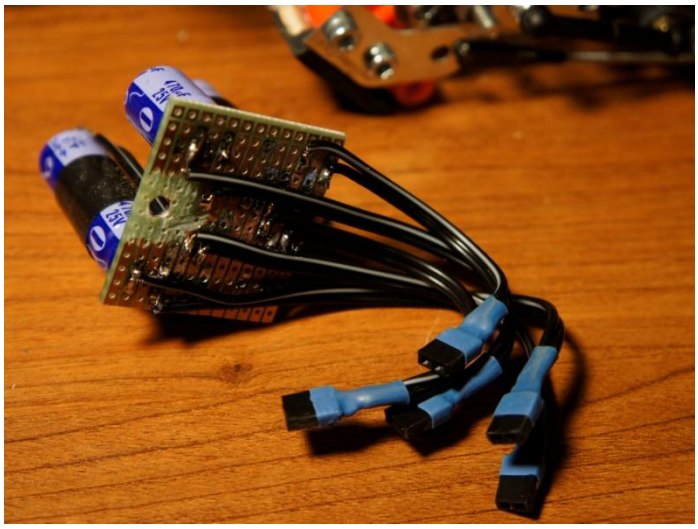


Rear Step-down converter assembly

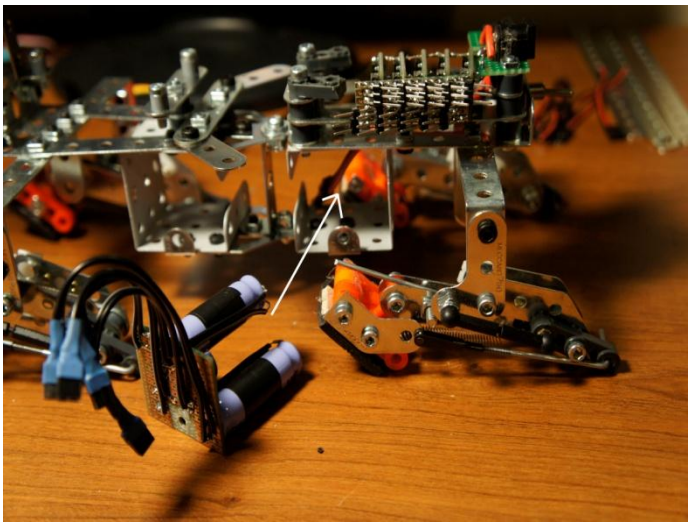


Installed into frame

Large 470uF capacitors are connected in parallel with the step-down converters to supply a steady current when the large inductive spikes from the servo motors occur. The size of these capacitors have been calculated to restrict inductive voltage spikes to less than 0.5V, plus or minus.



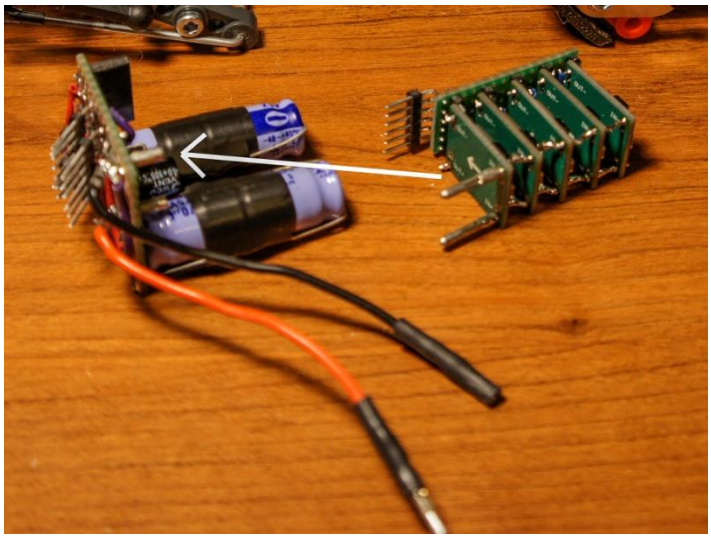
(5x) 470uF Electrolytic Capacitors



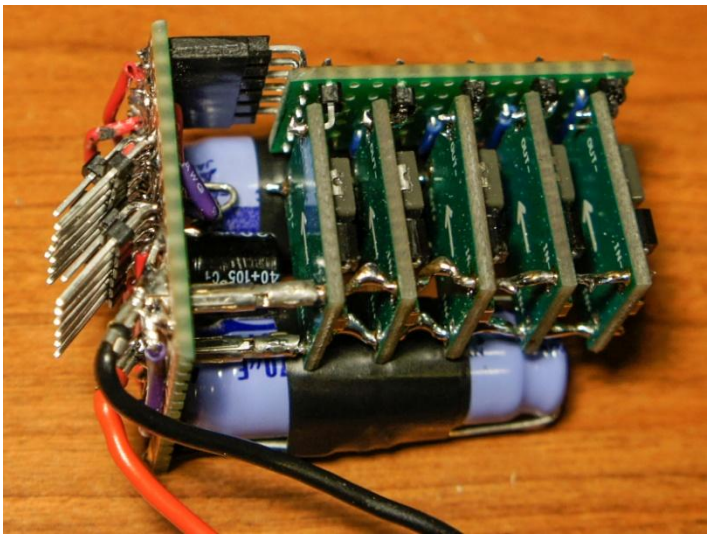
Installation

2.5 Front Power Grid

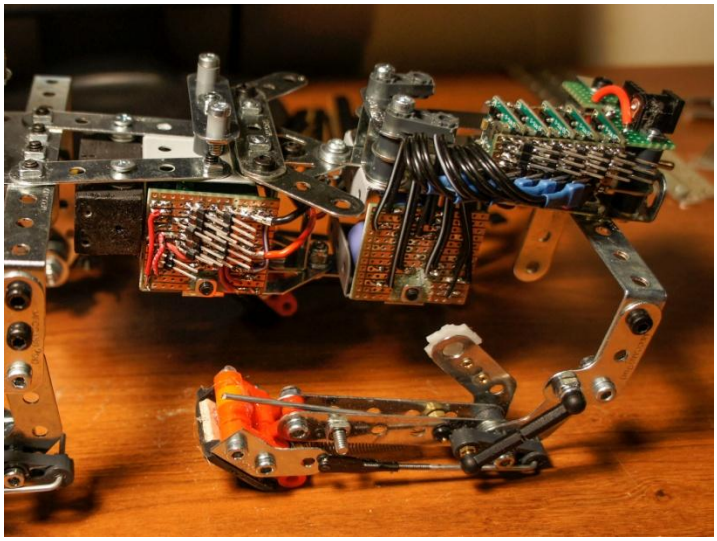
Similar to the rear power grid, the front power grid also uses step-down converters coupled with 470uF capacitor to supply a smooth current flow to each servo. Unlike the rear, all the components have been packaged together into a cube that fits inside of a frame cage. The 5th step-down converter supplies the microcontroller and peripherals with a steady 5V power supply. That output is coupled to a much smaller capacitor and a diode to prevent USB power from feeding back into the power system.



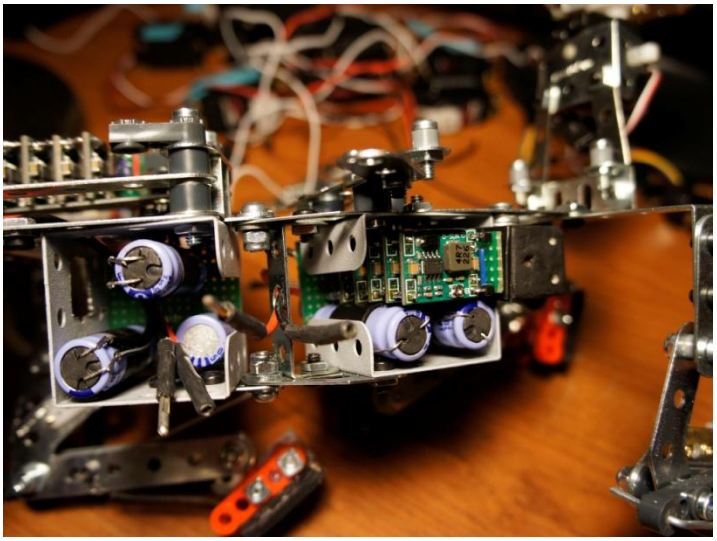
Component Assembly



Final Assembly



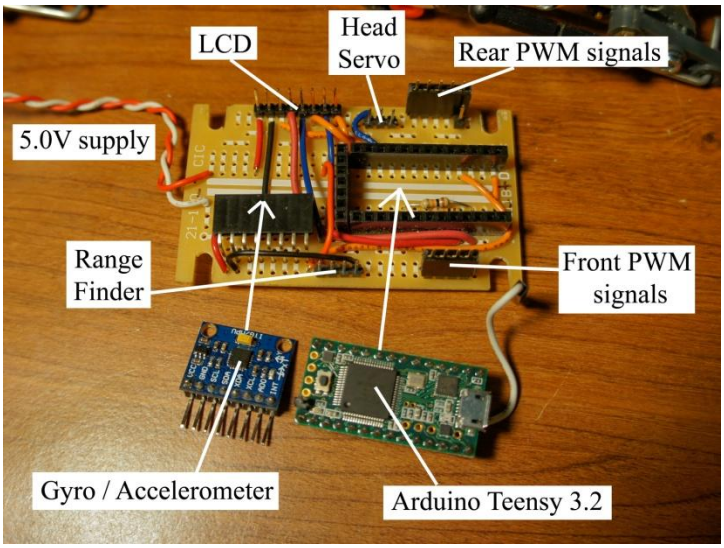
View From Left



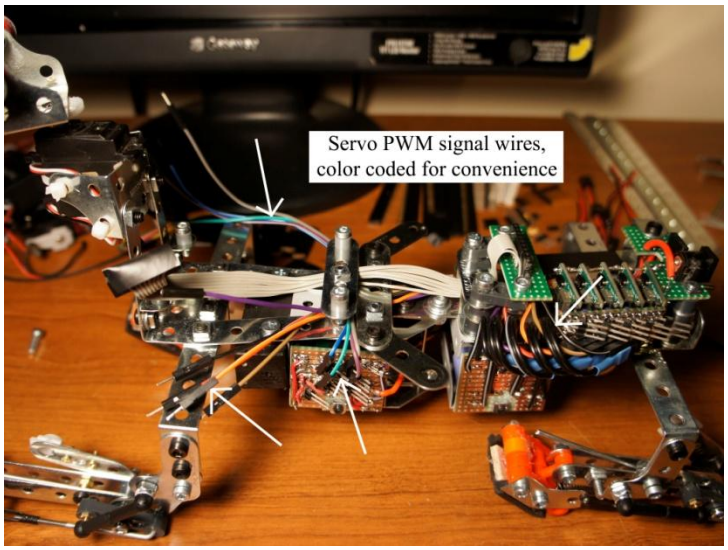
View From Right

2.6 System Board

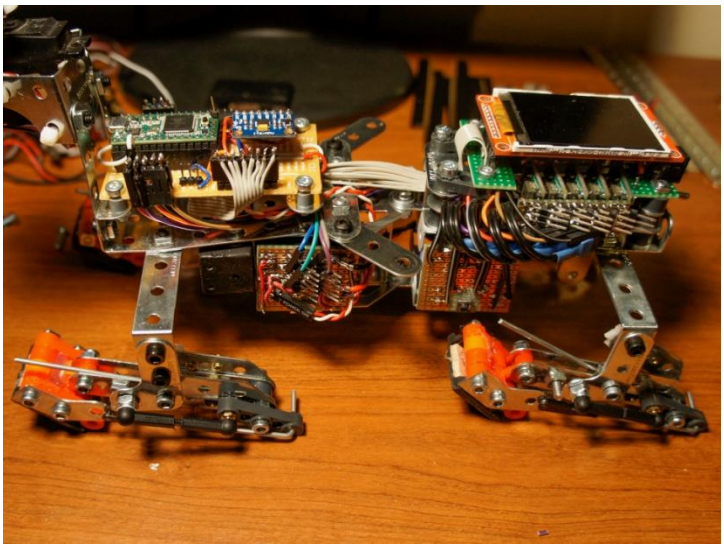
The System Board was designed to allow easy connections with external devices, and to fit within the size limits of the frame. All pins and connectors were soldered underneath to match the system schematic. Servo signal wires are color coded for identification, and a partial strip of grey IDE cable was used to connect the LCD screen. A diode allows the System Board to be powered from the on-board power grid, or from USB power.



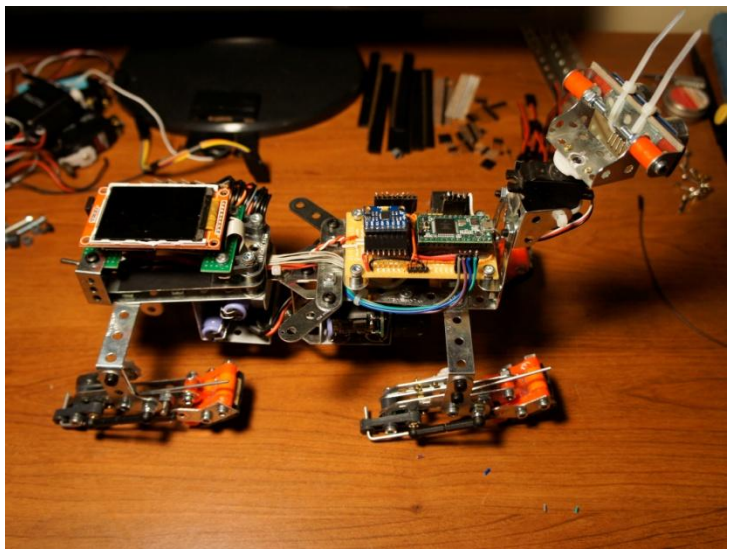
Component Connections



Output Wires



View From Left

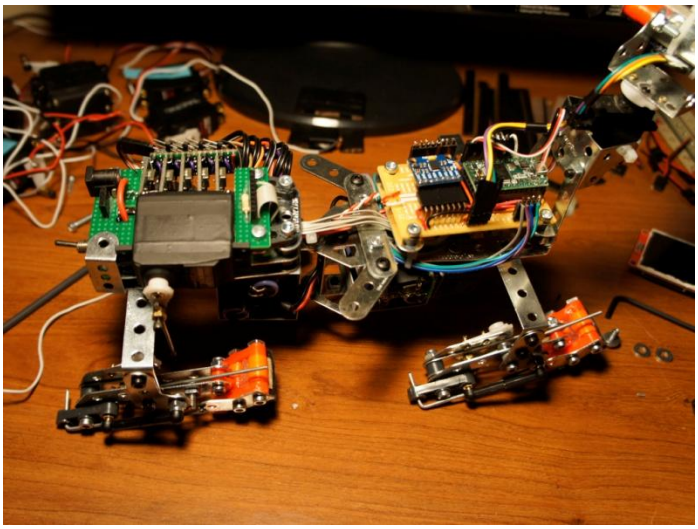


View From Right

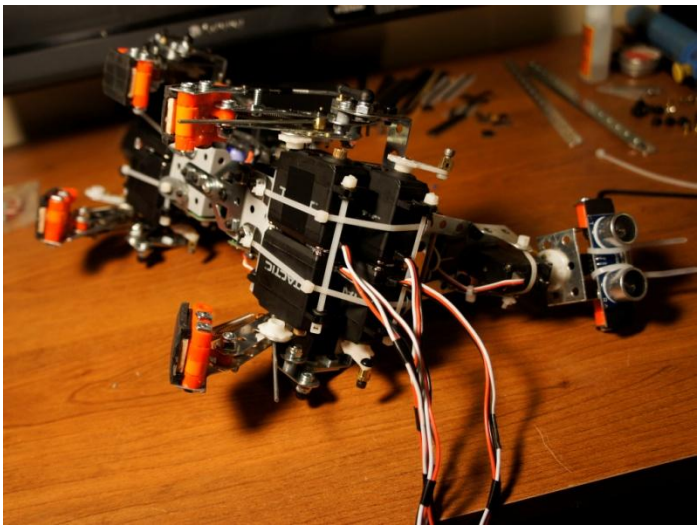
2.7 Servos

Tactic TSX55 Ball-Bearing Servo		
Volts	Torque	Speed
4.8	181 oz-in	0.24 sec/60°
6.0	203 oz-in	0.21 sec/60°
Dimensions		Weight
1.6" x 0.8" x 1.5"		2.1 oz (58 g)

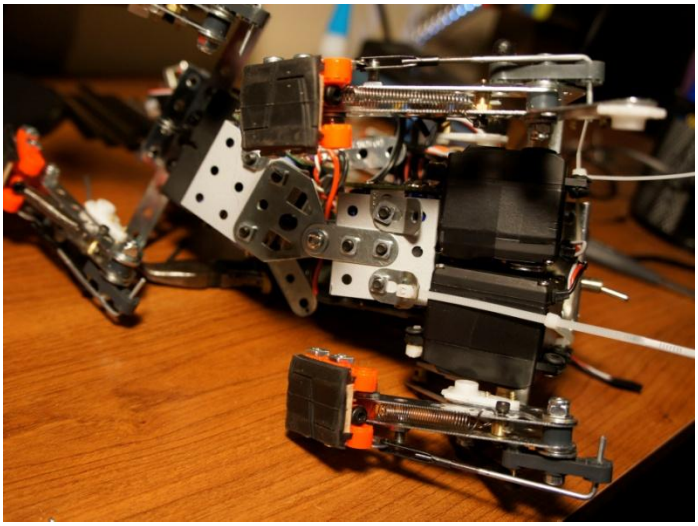
After doing some research and shopping, I found the best servo for this project to be the Tactic TSX55. They were each under $20 and provided the large amount of torque required to lift the robot. The servos all function using a PWM signal, with 1.5us being center of travel. The TSX55 servos are high-torque, heavy duty, metal gear, ball bearing servos capable of producing 203 oz/in of torque at full voltage. This is enough to support the weight of all the hardware while providing extra power to support an additional payload. The leg servos are secured with screws and zip-ties for easy removal, and the waist servo is held down by a metal strap. Lastly, a small generic mini-servo is used to rotate the range detector. It only delivers around 20 oz-in of torque, which consumes less than 0.5A at 5V, and is powered directly from the system board power supply.



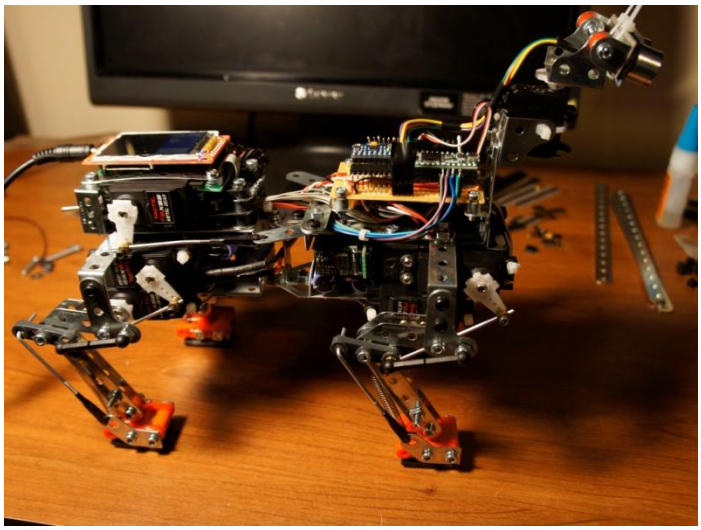
Waist Servo Installation



Front Servo Installation



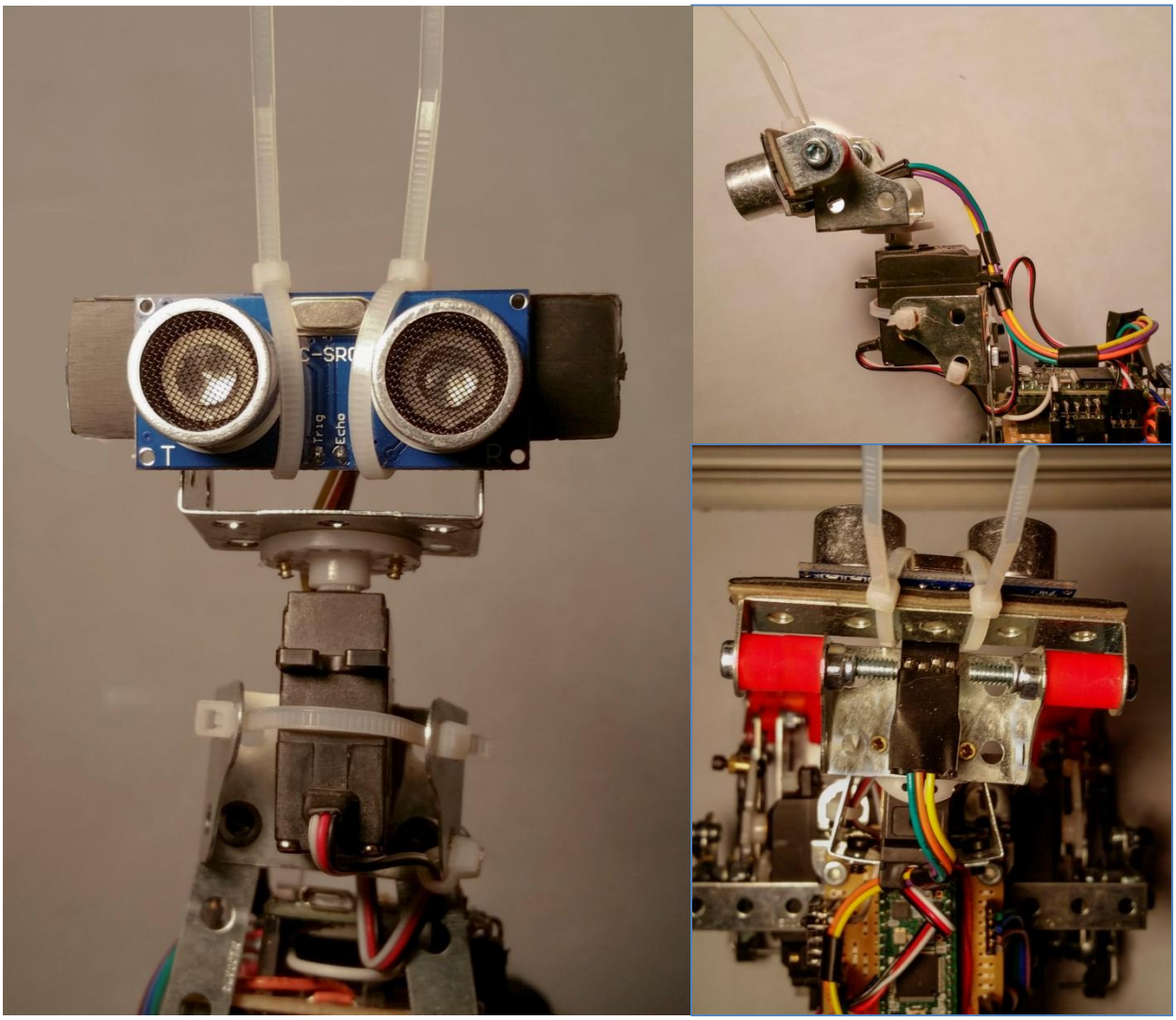
Rear Servo Installation



Linkages Calibrated

2.8 Radar

The head unit was constructed solely as a radar platform. I used an HC-SR04 Ultrasonic Ranging Module for the range detection, which accurately detects ranges between 1 inch and 13 feet. This module is mounted to an electrically insulated plate using zip-ties, and the mounting plate can be manually adjusted up and down. This assembly is mounted directly to a servo head with small screws, to allow precise scanning of the environment. All radar components are powered directly from the System Board power supply, so the system can operate when only plugged into USB power.



Radar Assembly

3 Software

3.1 Leg Geometry

The leg hardware for the servos have been carefully tuned and measured for each leg. Exact distances between key points of the movable legs are essential for geometric calculations to be performed. The Law of Cosines is used for determining the appropriate servo angles. First, the origin of the coordinate system is placed exactly at the shoulder joint, and the target (X,Y) coordinates are relative of that. It can then solve the distance and angle of the line from the origin to the point (X,Y), giving both the length of the third side of the red triangle and also its rotation. It can then use the Law of Cosines to calculate the angle of the corner that touches the servo, and subtract the red triangle's degree of rotation to determine our servo angle. Once it solves the angle of the bottom servo, it can calculate all points along the red triangle, such as the point where the green triangle meets the leg. With that, it can then calculate the distance between that point and the top servo, solving the length of the third side of the green triangle. Using the Law of Cosines again, it can solve for the angle of the top servo.

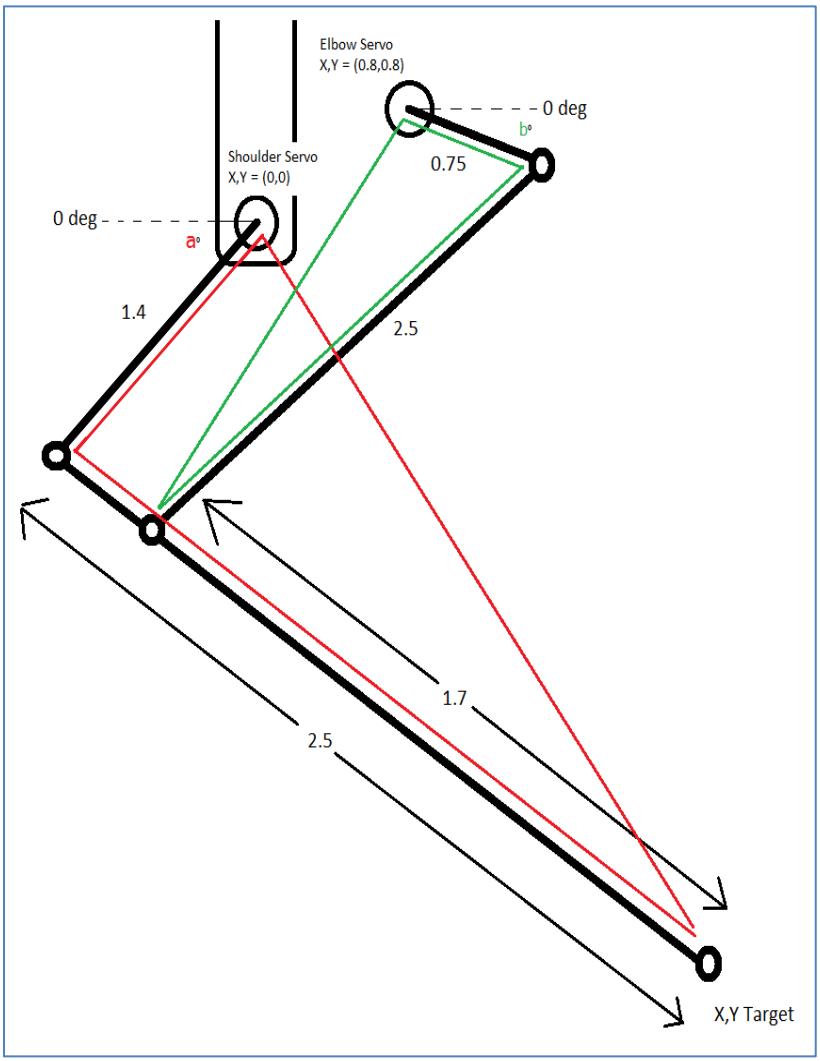


Figure 4: Leg Geometry

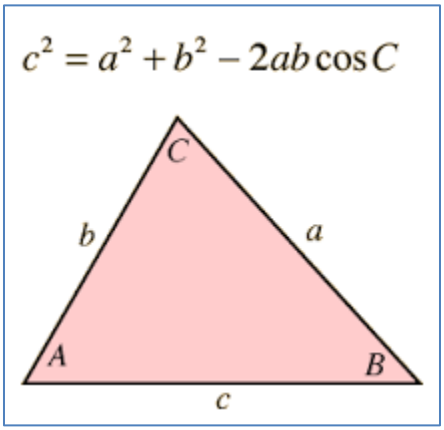


Figure 5: Law of Cosines

3.2 Pivot Compensation

As the frame pivots, the feet naturally rotate with it. This causes a skewed placement of the feet which will disrupt the center of balance. In this design, the feet automatically compensate their horizontal positions to maintain a square base. The entire range of motion for each foot is offset by this amount to allow smooth transitions as the frame pivots left and right.

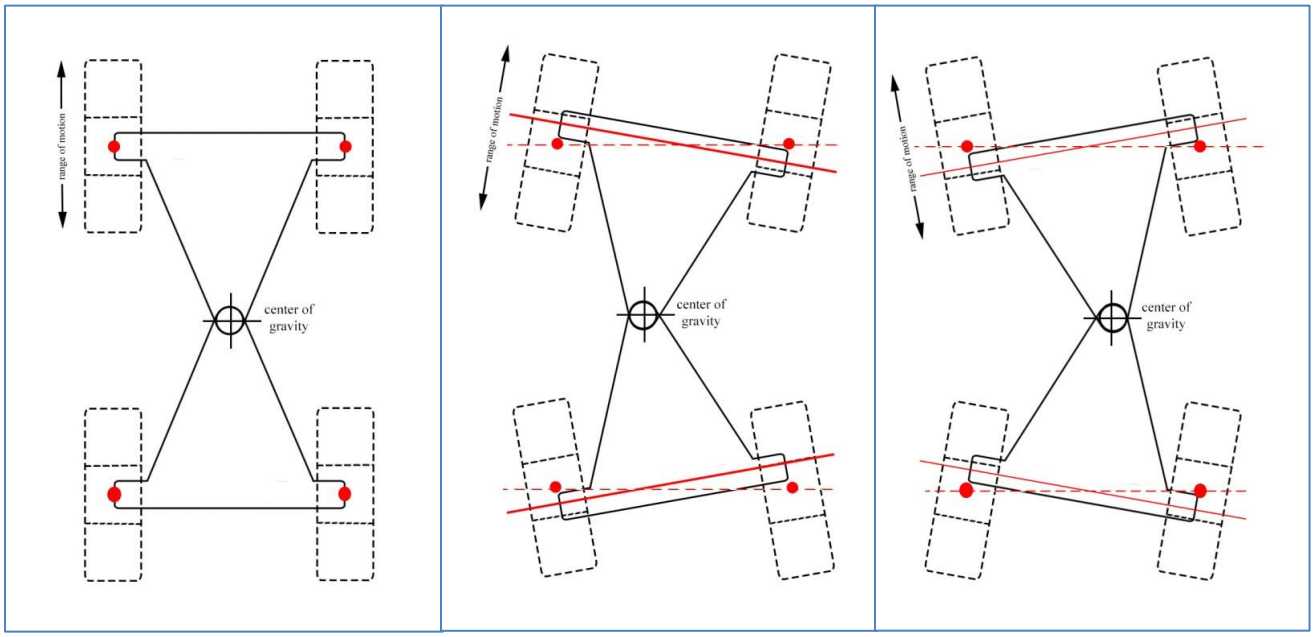
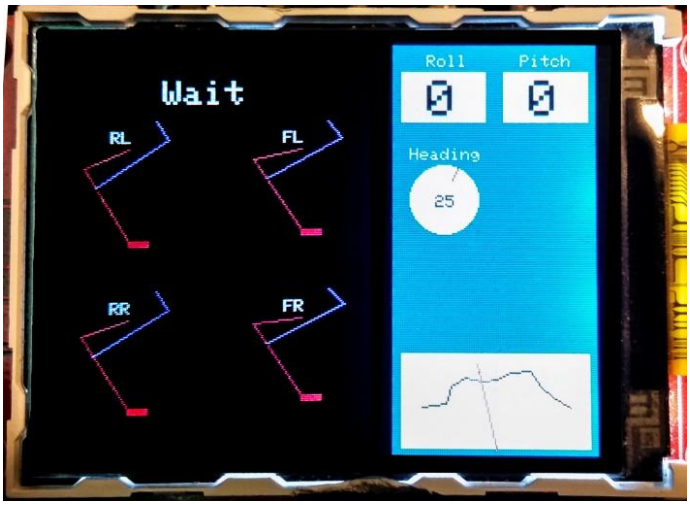


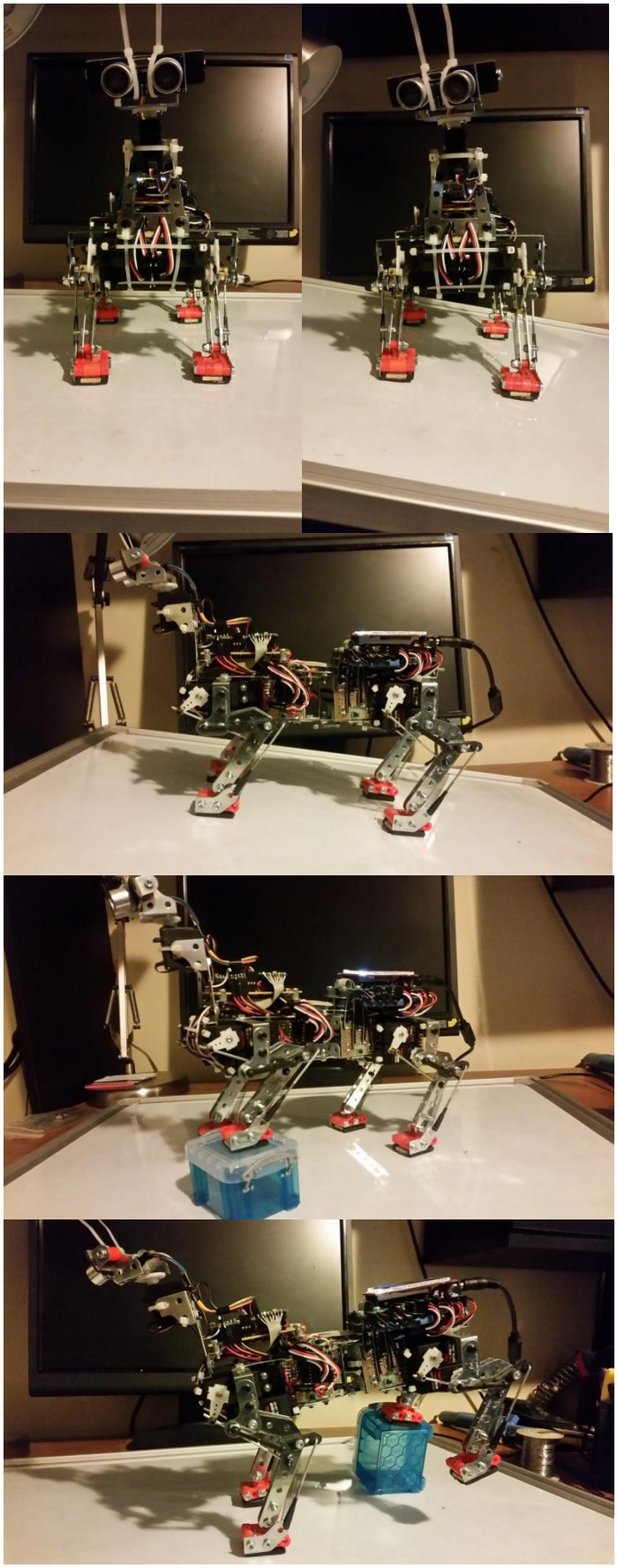
Figure 6: Pivot compensation

3.3 Graphics

The graphics display shows the current status of the robot. The four leg positions are drawn to scale and show the current calculated positions of each servo. The command being executed is displayed at the top, above the legs. The Roll and Pitch that is used to maintain stability are both displayed in real time, as well as the predicted yaw heading. The radar map is kept in the bottom corner and is a representation of the distances found by the range finder. The orientation of the head is also synchronized with a red sweeping line inside the radar map.



3.4 Self Leveling



This robot can automatically compensate the height of each leg to maintain a level orientation. Roll and pitch are independently monitored and adjustments are made for each two-dimensional plane. The robot will slowly adjust all four legs, for each axis, in an attempt to bring roll and pitch values back to 0. To maintain symmetry, legs are adjusted in pairs and are moved opposite of each other to maintain the same average height. Limits are placed on these adjustments to stay within the hardware limit of travel.

The speed at which this system responds can cause extremely unstable conditions. If the reaction speed is too fast, the legs will overshoot their target, and the extra momentum can result in an unrecoverable side-to-side oscillation. By using multiple equations, it can react differently for each range of roll and pitch values. This allows it to make quick gross adjustments, while still reducing the speed gracefully as it reaches zero. Large roll angles are defined to detect a roll-over, and to avoid overreactions near the tipping point.

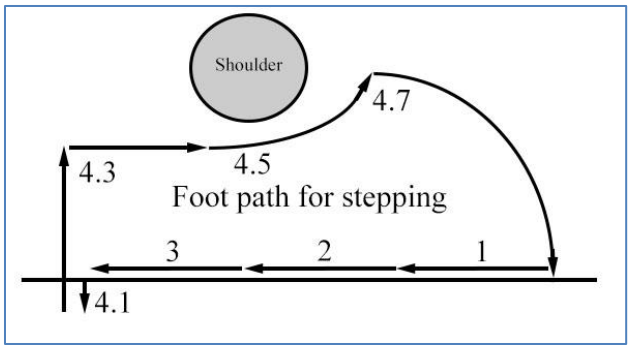
The sensor data obtained from both the accelerometer and gyroscope are used for a quaternion algorithm which computes them as two vectors in a three-dimensional space. While in motion, this algorithm produces estimated values for yaw, pitch, and roll. While the robot is steady, gravity is used to calibrate center.

Once calibrated, this system performed well on both a tilt-table test, and an obstacle test. The legs have a good natural reaction to large events, and constant adjustments can keep it level while in motion.

Video: https://github.com/alextweedy80/Projects/blob/master/Robot/Autolevel.avi?raw=true

3.5 Walking

Figure 7: Step path



The walking algorithm was designed to keep the center of gravity balanced within the 3 planted feet. To create distinct phases of a step loop, it is divided into 4 segments. The range of motion of the feet are divided into 3 even segments and the 4th segment is a lift and place routine, as shown in Figure 7. Each foot is always in a unique phase so that only one foot is stepping at a time, and stepping is alternated from front to rear, one side at a time. As each side is stepping, the frame pivots away to adjust the center of gravity away from the lifted foot. The auto-leveling routine is also sent configuration variables that cause it to roll and pitch away from the lifted foot. This assists by preemptively orientating the robot away from falling when the foot is first picked up. To further assist this function, each foot quickly presses down before lifting up, which causes the weight to securely rest on the red triangles shown in Figure 8.

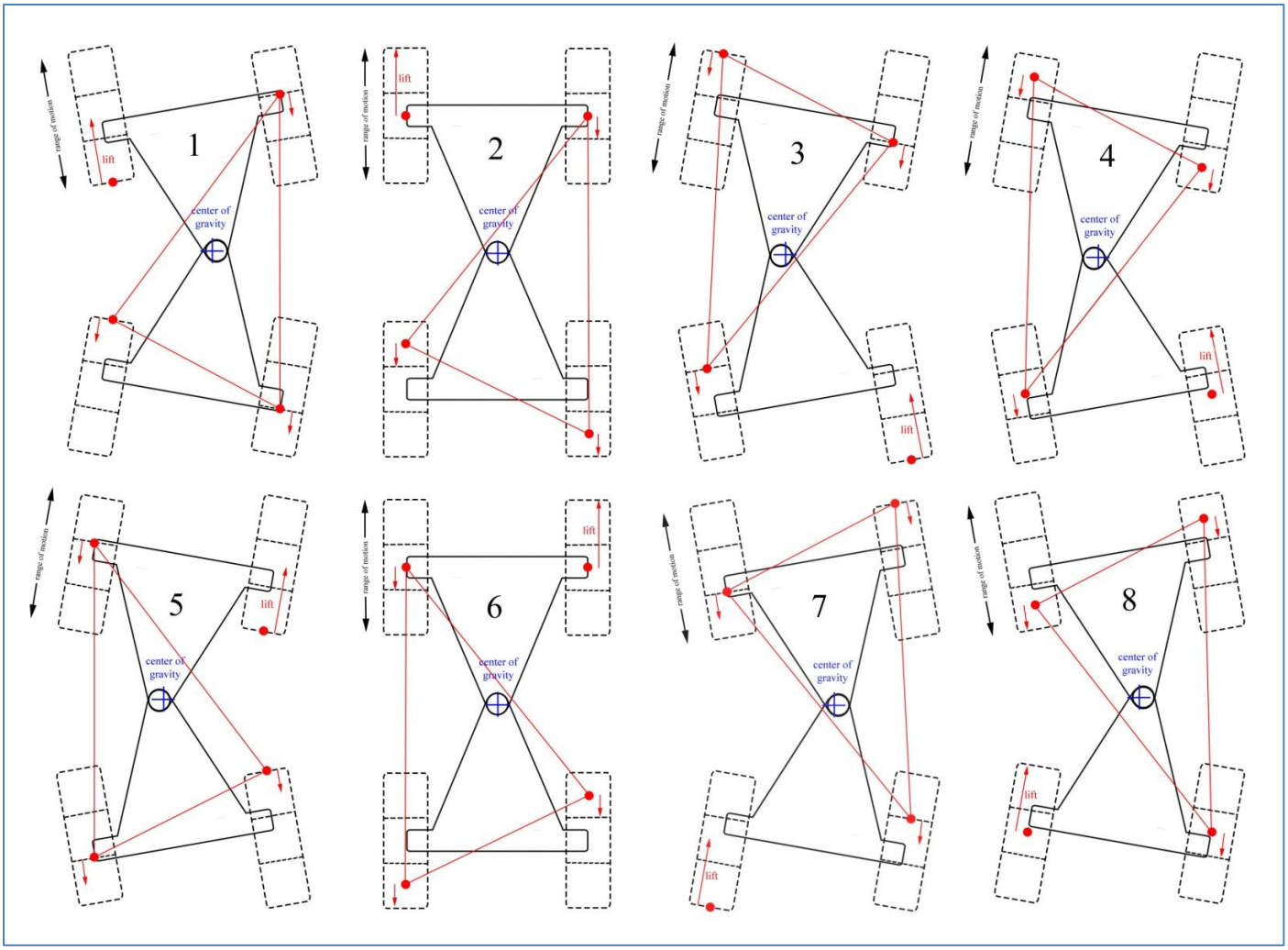


Figure 8: Walking Algorithm

Video: https://github.com/alextweedy80/Projects/blob/master/Robot/Walking.avi?raw=true

3.6 Radar

The radar is used to detect objects that are in front of the robot. The resolution is adjustable in the source code, and the range of motion was calibrated to the servo. These variables are used to accurately display the radar map using the correct angles. As the range finder sweeps across the field, the distances are stored in an array, and a new line segment is drawn on the radar map. This system was designed to build a 2D map as it moved through the environment, but development is still in progress.

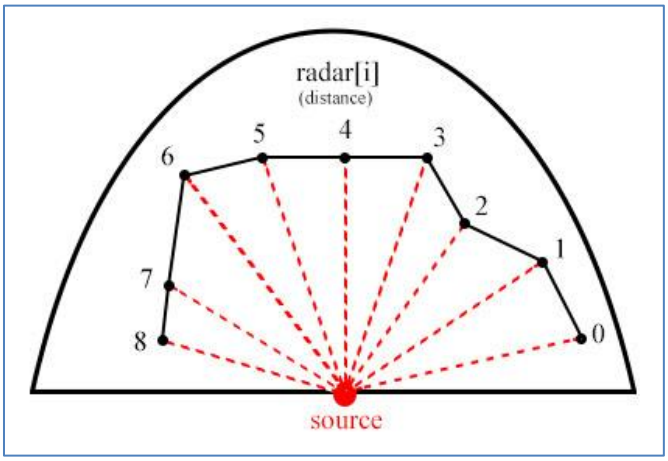


Figure 9: Radar Map

The radar system operates by rotating the range finder to each point, in a back and forth sweeping motion, and activating the range finder at each one. To activate the radar pulse, it sets the Trigger line high for 10us, and then measure the time until the falling edge occurs on the Echo line. It uses a falling-edge triggered hardware interrupt on the Echo line to maintain high precision. This "ping" is repeated multiple times at each position to develop an average value to store in the array.

3.7 Current Development

Bluetooth Connection:

Currently, commands must be entered into the source code before uploading. By adding a Bluetooth module, more sophisticated controls can be developed. Development of an Android application to control the robot is in progress. Bluetooth could also allow access to a network of various remote devices to help direct commands, including the internet. The hardware component will be added underneath the Gyro/Accel module and will also communicate on the SPI bus.

Steering:

To allow the robot to walk around obstacles, a turning algorithm must still be developed. While walking, the center of gravity restricts large variations from the normal algorithm. I believe a modified walking algorithm that involves slipping the feet, pivoting the frame, and forcing a heavy roll can maintain center balance while turning.

Radar Based Route:

Although the radar is collecting data and drawing a map, it is not affecting the actions of the robot. Once steering has been developed, the radar values can be used to locate open routes and avoid obstacles. The (X,Y) coordinate system of the feet are calibrated in inches, so real distances can be used to build layers of data into a 2D map.

3.8 Software Outline

setup ()

- load startup routine into memory
- initialize servo pins
- set PWM frequency
- initialize range finder
- initialize LCD screen
- initialize gyro/accel
- set head, legs, and waist to start position

Main loop()

- processGyro()
- loadCommand()
- executeCommand()
- autolevel()
- moveServo()
- sweep()
- drawData()

- calculate yaw, pitch, and roll
- load next command into memory?
- load next movement of current command into memory?
- adjust height of each leg to maintain level
- adjust each servo to move feet to new locations
- run radar routine to build radar map
- draw graphics to display

Figure 8: Main Loop Description

processGyro()

- Capture and adjust current gyro/accel values
- Run Quaternion algorithm to calculate Yaw, Pitch, and Roll

loadCommand()

- Check flag to verify the current command has completed
- load movements for next command into memory

executeCommand()

- Verify the current movement has completed
- load next movement into memory to be executed by moveServo()
- If entire command is complete, set flag to trigger loadCommand()

autolevel()

- Calculate new pitch and roll values from locally stored gyro/accel data
- Use various equations to calculate amount of leg height compensation to adjust
- Verify final values are within physical limits

moveServo()

- Calculate the progress % of time elapsed / total time movement should take
- Use preset equations and progress % to determine where each foot should be
- Calculate foot compensation for any twisting at waist so feet remain parallel.
- set waist servo to new position
- call moveArm(leg, x, y) for each leg

moveArm(leg, x, y)

- Use law of cosines to calculate angle of bottom servo
- Use law of cosines to calculate angle of top servo
- Draw leg position on LCD screen
- Convert angle into PWM value
- Output PWM signal to servo

sweep()

- send head servo to new position
- wait for servo movement to complete, based on time
- trigger radar pulse
- wait for radar return and then store value
- repeat radar ping to gather an average value
- store final value and update radar map on LCD

Interrupt Routines:

pulseEnd()

- subtract current time from start time to compute elapsed time of radar return
- set dataReady flag to true

readMPU()

- new gyro/accel data is ready, copy values from registers into local variables

Source Code: https://github.com/alextweedy80/Projects/blob/master/Robot/RobotWalking_v3.ino