Work Breakdown Structure:
Search & Rescue Robotic Quadruped

SACRAMENTO
STATE

10/21/2019

Team 9

Alfred Martinez III, Kristian Ornelas, Edgar Granados, Marcus Huston

Professor Tatro – Professor Levine

Elevator Pitch:

We are designing a robotic quadruped that will be able to operate semi-autonomously to aid in search and rescue operations.

Executive Summary:

This document outlines the different sections of the project and attempts to place a reasonable timeline of events for them to be completed. Building a robot is a huge undertaking, and requires many subparts working accurately and simultaneously. To this aim, the project has been split into categories based on the phase of the project. These included Mathematical Modeling and Prototyping Phase, Motor/ Leg Control System Phase, PID Control System Phase, and Visual Sensor Phase. Each will be worked by different team members to ensure that the timeline of the project and the desired results do not go passed the time that has been allotted.

Search & Rescue Robotic Quadruped

Alfred Martinez III, Kristian Ornelas, Edgar Granados, Marcus Huston
Department of Electrical & Electronic Engineering, Computer Engineering Program, CSU Sacramento
6000 J Street, Sacramento, CA, USA

***Abstract*— This document is a breakdown of the necessary work needed to be completed in order to implement the features required in the search and rescue robotic dog. Looking from the top level down, a robotic dog requires dynamic locomotion, a central control system, and an array of visual sensors to work in real world environments. These tasks will be the main effort of this design project and will be outlined throughout the document.**

***Index Terms*— Robotics, Quadruped, Search and Rescue, Disaster Worker, Robot, Open Source, ROS, DC Motor, Semi-Autonomous, Machine Vision, PID Control**

I. Introduction

It is extremely important for a project to have a timeline in order to ensure its methodical completion and for one to objectively look at the status of the project and determine the percentage of completion. This document is a breakdown of the work required to be completed in order to meet the measurable metrics and features.

II. Accurate Leg Position & Control

1.1 Ability to rotate the motor at a predetermined RPM

The ability to rotate the DC motors at a specified RPM is a simple yet critical required feature as it will allow us to control the robot in the manner that we want and its response to user requests.

1.1.1 Encoder Data from each motor must report accurately and with consistency

The first step into achieving this task will be to interface the encoders with the ODrive Motor Controller. Each motor(x12) will have an encoder attached to it and that encoder will need to interface with an ODrive. The ODrive supports AB, ABI, or SPI communication; so as long as the encoder we choose supports any of these communication protocols, the ODrive will be able to read data from the encoders. In addition, the ODrive website documentation has information on calibrating the encoders. We must calibrate the encoders so that they are accurately outputting the motor's position and speed. Luckily, the ODrive comes with built in software tools that we can run in order to calibrate the encoders. Specifically, all we really need to do is have an ODrive with a motor and encoder connected to their respected pins, and run a python program(odrivetool.py) that will automatically calibrate the encoders for us. As for the time required to complete this, we estimate that this should take us no more than 1 week to complete. That is, have the encoder interfacing with an ODrive; and transmitting data over.

1.1.2. Encoders are able to accurately output readings of motors at speeds up to 3000 RPM with no load

The encoders we decide on will need to be able to accurately output data for motors rotating at around 3000 RPM. The motors we will be using for this project are called the Turnigy Aerodrive SK3. Their rated velocity is rated at 192 Kv, meaning that if the motor was powered at 1V, they would spin at 192 rpm with no load. For our project, if each motor was powered by a 15 V voltage source, then the total rated velocity that the motors would spin would be 2880 rpm (192 Kv * 15V). The encoders we may use for this project are known as the 8192 CPR AMT102 and they are able to read motor speeds of up to 7500 rpm; two and half times that of our motor's rpm. The ODrive motor controller has a

maximum voltage rating of 24V, so even if we powered the motors at the ODrive's maximum voltage, the maximum speed that could be reached by the motors would be 4608 rpm (192 Kv * 24V). Thus, the encoders would be more than adequate to handle such speed readings. This task wouldn't require much programming or setting up as we only need to choose the proper encoder that would be meet the minimum speed requirements. Therefore, the only thing that this task would require would be choosing the encoder.

1.1.3. ODRIVE interface with microcontroller using SPI or I2C or UART to send data to master device(s)(ODrives)

In order to complete this task, the first step would be to have an MCU programmed and running as the "synchronizer" between the IMU sensors and the ODrives. Our optimized control system would use a simple MCU like an STM32F303K8 Nucleo, which would be programmed as the master device that would transmit readings from 2 IMUs, to each ODrive at the same time. The entire control system would use the SPI Communication protocol, running at over 1 MHz. Each ODrive would receive the same readings from the IMUs simultaneously and perform the appropriate calculations based on their local parameters i.e. the positions of each motor they are controlling. In order to accomplish this task, the very first thing that needs to be done is to set up an MCU as a master device that would take readings and transmit them over using SPI. This is a very trivial step if it is done using ST Microelecontric's supported IDE known as STM32CubeMX, as all we need to do is assign certain pins to the SPI clock, MOSI, MISO, and GND. The CS (Chip Select) pin would not be used in this design, as all ODrives would receive the same IMU readings, and they would perform their own "local" calculations. The part that would require more work and time to complete would be modifying the ODrive firmware so that it uses SPI as its default communication protocol; and also using its built in FREERTOS middleware, in order to assign tasks and implement a PID controller. By default, the ODrive utilizes UART communication to communicate with external microcontrollers. However, the ODrives have a STM32F4 chip that is used to program each of the pins on its board. Therefore, we would be able to set up the ODrive in the same manner we set up our Nucleo using the STM32CubeMX IDE. This step is **CRUCIAL** to have completed as it is a prerequisite to other tasks such as implementing a PID controller, Static Stability, Dynamic Stability, and Gait Transitioning. Therefore, if we can't have the IMUs, MCU, and ODrives interfacing with one another, we will be unable to move forward with other tasks that will be necessary to have a moving, stable robot. With that being said, we estimate that this task will require a great portion of our time to complete, and we will assign it a completion time of 4-5 weeks.

1.2 Ability to rotate leg sections to specified angle

1.2.1 Motor must be able to transmit its RPM to pulley system and then to motor with minimal loss through physical transmission.

After we can accurately get data from the encoders to the ODRIVE, we will need to code specific values to test the motors RPM. Once we get accurate RPM readings that match our code, we can begin to test the motors with use of the ball screws. This will mean that the motors will be able to speed up or slow down based on the specific RPM we have predetermined. Therefore, we will need to test and verify that there is no slip while the motors are spinning and that the ball screw moves up and down to the predetermined spots. This will require a lot of tests in order to verify that the motors rotate quick enough as well as move the ball screws fast. If the motors and ball screw combination are not accurate, it will make other tasks such as walking and stabilization almost impossible.

1.2.2. Motors must remain in the operating temperature range to prevent overheating and meltdown

Once we have test data, accurate RPM speed, and the specified ball screw movements, we will test them and put them under load to test their accuracy. This will mean that we will need to measure the temperature of the motors and that they stay within the specified ranges without overheating or malfunctioning. If the motors show too much heat, that can cause the robotic quadruped to malfunction or destroy other parts on the robot. We need to

verify that the motors stay within a reasonable temperature under load, so we keep the project safe and don't harm any other components.

1.2.3. Specified leg position must be within 10 degrees

After measuring and matching the data from the encoders to the ODRIVE, checking the accuracy of the motors and ball screws, we can then start testing and coding to make the leg move. This step will only be able to happen with accuracy if we are getting the correct data and communication between the encoders and the ODRIVE as well as the ball screws to move to accurate locations. We will then have to test and measure how much angle we can get from the hip, thigh, and knee of the robotic quadruped. With the test and measuring, we will get enough data to be able to code the robotic quadruped leg to be able to move to a specific given value of degrees to move to and be able to be within 10 degrees. This can be measured by someone telling it to go to a specific value and using a protractor to measure that the specific angle is correct.

III. Statically Stable Body

2.1. Ability to maintain three points of contact while moving

In order to maintain static stability, we will need to have a feedback system that will adjust the outputs; in this case, the motor speed based on the input data, IMUs.

2.1.1. Calibrating the IMUs

All IMUs require some form of calibration to output the proper readings. We will need to determine what offset the two IMUs are outputting and subtract this offset reading in order to have accurate data. To do this, we will need to place our IMUs on the stationary robot. The robot must be stationary and in balanced form as this will be the origin point the PID controller will try to maintain when performing calculations. We will write some simple instructions in C that will collect IMU readings for a small amount of time. We will then find the average of these readings and subtract this average from the offset readings of the IMUs. This will give output readings of or very close to 0 meaning, that the current position is set to be our origin position; or balance point.

2.1.2. PID Controller Setup

Once the IMUs are calibrated, the next task will be to implement a PID controller in C. The PID controller works as a closed loop feedback system that adjusts the output based on the inputs and some output fed back to the input. This mathematical model will be coded within our ODrives and the output will be the varying speed and position of the motors, based on the IMU readings being inputted via SPI communication. There are various code implementations of a PID controller online; our main task here will be to figure out how fast the PID system will need to adjust its outputs and how to modify the desired point it is trying to achieve. For instance, if the robot is standing on an incline, how can we adjust the desired point so that the PID controller is trying to maintain this desired point.

2.1.3. Directly Coded Gaits

Once the PID controller is implemented on each ODrive; we will then need to code the appropriate gait transitions. As the robot begins to move one limb from its center of mass, there must be a reaction from another leg, so that it is able to maintain its balance. This will be implemented in code, we will primarily use the IMU inputs to initiate a state change once the center of mass of the robot has gone over a fixed threshold. To accomplish this task, we will need to determine what the threshold change in the IMU reading is and initiate a movement from a leg that will counteract the change in center of mass from the first leg; to balance out the robot.

IV. Dynamically Stable Body

3.1. Ability to balance using IMU

3.1.1. Measure self-balancing time to keep the body level with ground

Making the robot able to balance itself on uneven terrain will include developing a feedback system that can recognize the differences in terrain and the imbalances in the body of the robot. To this aim, IMU sensors will be implemented in the main carriage of the robot and will feed into the robot's main motor control source code. This will be split up into a research phase, prototyping phase, and a full-sized model phase for the final design. This semester, the group is hoping to have a working 3D modeled miniature model of the quadruped with the IMUs in use.

3.1.2. Demonstrate PID code that is presentable in a block diagram (Mathematical Model)

While working on the self-balancing controls of the miniature model the intention is to create a working MATLAB Model for the final model of the robotic dog to be implemented in the Spring semester. To do this, the Electrical members of the team will be working with the Mechanical members of the team to develop a 3D model in Simulink. This model will measure torque and weight requirements with adjustable limb and body weight, current, voltage, and angular movement.

3.1.3. IMU SPI Protocol 1 MHz frequency output with MCU

The team is working to make one of their measurable metrics a 1 MHz data reading from the IMU using SPI communication. This will be a useful practice to achieve the final desired results next semester of having a large series of data streaming through SPI protocol on the final version of the robot. This is being conducted by the Computer Engineering majors. The plan is to adjust the current firmware aimed at Arduino Boards to fit a pure M4 Cortex Processor like the ones used on the final robot.

3.1.4. IMU Accuracy within 10 degrees

This step will be conducted after the IMUs are running and producing data in a computer terminal. Here, the measured results from an external device will be compared to the readings printed in the terminal. The IMUs will be tested until the data is within a 10 degrees margin of error to ensure the data is accurate enough for the final robot to be dynamically stable at its higher weight.

3.2 PID for Large Model

3.2.1. MATLAB Model of Dynamic Motion

One key piece being worked on throughout the first semester is the MATLAB Simulations of the robot. This will be important for finding small errors in balance and movement of the dog especially over uneven terrain. This will be broken up between the controls part of the group who plan on mapping the range of motion and linear algebra for the kinematics. That way it can be directly translated into code from the MATLAB Model.

V. Visual Sensor/Controls

4.1. XBOX ONE Kinect Setup

The XBOX ONE Kinect in this project will act as the eyes of the quadruped and will utilize machine vision and machine learning concepts. The Kinect was chosen over other cameras because of its cost and its online community support.

4.1.1. Interface the XBOX Kinect with microcomputer and send output

The Kinect will need to interface with the Nvidia Jetson Nano and will need to display the Kinects output in order to showcase the possibility of data transfer between devices. This step is a prerequisite for the remaining sections of the XBOX ONE Kinect setup.

4.1.2. Map the terrain in front of the Kinect in Robot Operating System (ROS

The Kinect will also need to perform 3D mapping using the Robot Operating System which already has built nodes capable of Simultaneous Localization and Mapping (SLAM). The point of view of the Kinect is limited to roughly 55 degrees horizontally and 45 degrees vertically with a max range of about 20 feet.

4.1.3. Detect object size up to 15 feet away

A very important feature of the robot is to detect objects for object avoidance. The Jetson Nano in conjunction with the Kinect need to be able to detect objects at a range of 15 feet away using ROS.

4.1.4 Use the Kinect to recognize walls/stairs/furniture

Once the Kinect and Nano are able to detect objections, they also need to be able to recognize the difference between walls, stairs, and furniture. The reason this is necessary is for the future iterations of this project if the designers decide that they want the quadruped to maneuver stairs and obviously for object avoidance.

4.1.5. Using 3D mapping data as input variables for robot decision-making

The data being produced by the Kinect will need to be processed and interpreted by the IMU so that the ODrive controllers can make the necessary controls for decision-making. ROS should have the necessary node that would be able to publish or output the requested data but the Jetson Nano also has GPIO pins for direct interfacing. The 3D mapping data will be in the form of point clouds so the IMU will need to receive this specific data via a communication protocol, not yet specified. Once the IMU interprets the data, the ODrive controllers will issue DC motor commands, thus maneuvering the robot.

4.2. Live Stream Remote Feed & Controls

This feature is a necessary requirement for the quadruped to be semi-autonomous. The robot is not fully autonomous so the user will need to be able to provide commands and directions for the robot to complete.

4.2.1. Live stream output to an Octomap/Point-Cloud

The Jetson Nano will need to livestream the data being produced by the Kinect to a Graphical User Interface (GUI) The GUI can be created in many ways but must provide the user a first-person point-of-view of the robot's current position. The GUI should also be configured for further development and improvement for the next section's requirements.

4.2.2. Send remote control commands to Quadruped

The GUI written in the previous step is a prerequisite. This step is fairly open-ended and is up to the designer of what route should be taken. The robot can be commanded via the GUI or a separate controller, such as a video game controller. If the designer decided to control the robot via the GUI, the interface will need buttons used as controls for the robot and should give vague instructions so that the user is not in complete control over the robot's motors. The controller would need to follow the same requirements but will require a different method of implementation.

VI. Conclusion

The importance of breaking down all the necessary work for a complex project with 8 engineers involved is insurmountable. The Search & Rescue Quadruped has its fair share of complexity and requires many new types of hardware, software and methods in order to implement the desired features. This project will need a diverse set of skills with consistency in following the timeline to avoid falling behind.

References

[1] Kalouche, Simon. "Design for 3D Agility and Virtual Compliance Using Proprioceptive Force Control in Dynamic Legged Robots." *The Robotics Institute Carnegie Mellon University*, Carnegie Mellon University, Aug. 2016, https://www.ri.cmu.edu/publications/design-for-3d-agility-and-virtual-compliance-using-proprioceptive-force-control-in-dynamic-legged-robots/.

[2] Boston Dynamics [Online]. Available: bostondynamics.com

[3] Introducing SpotMini. Boston Dynamics. https://www.youtube.com/watch?v=tf7IEVTDjng

[4] S. Seok, A. Wang, D. Otten, and S. Kim, Actuator design for high force proprioceptive control in fast legged locomotion, 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012.

[5] G. Kenneally, A. De, D. Koditschek. "Design Principles for a Family of Direct-Drive Legged Robots." IEEE ROBOTICS AND AUTOMATION LETTERS, VOL. 1, NO. 2, JULY 2016.

[6] Huang, W., Kim, J., and Atkeson, C. (2013). Energy-based optimal step planning for humanoids. In 2013 IEEE International Conference on Robotics and Automation (ICRA), pages 3124–3129, Karlsruhe, Germany

[7] Kuindersma, S., Permenter, F., and Tedrake, R. (2014). An efficiently solvable quadratic program for stabilizing dynamic locomotion. In IEEE Intl. Conf. on Robotics and Automation (ICRA), Hong Kong, China.

[8] Marc Sons, Christoph Stiller, "Efficient Multi-Drive Map Optimization towards Life-long Localization using Surround View", *Intelligent Transportation Systems (ITSC) 2018 21st International Conference on*, pp. 2671-2677, 2018.

[9] David Wooden, Matthew Malchano, Kevin Blankespoor, Andrew Howard, Alfred Rizzi, Marc Raibert, "Autonomous Navigation for BigDog", *IEEE International Conference on Robotics and Automation*.

[10] Joel Chestnutt, *Navigation Planning for Legged Robots*.

[11] Dr. Asad Yousuf, Mr. William Lehman, Dr. Mohamad A. Mustafa, Dr. Mir M Hayder, "Introducing Kinematics with Robot Operating System (ROS)", *122nd ASEE Annual Conference & Exposition*.

[12] Kenta Takaya, Toshinori Asai, Valeri Kroumov, Florentin Smarandache, "Simulation Environment for Mobile Robots Testing Using ROS and Gazebo", *20th International Conference on System Theory Control and Computing (ICSTCC)*, 2016.

[13] Ilya Afanasyev, Artur Sagitov, Evgeni Magid, "ROS- Based SLAM for a Gazebo-Simulated Mobile Robot in Image-Based 3D Model of Indoor Environment", *International Conference on Advanced Concepts for Intelligent Vision Systems*, 2015.