Balancing Robot: An investigation into Realtime Operating Systems for Dynamic Control

Operating Systems - CPE534-01

Michael Angeles

May 3, 2024

Michael Angeles
CPE534-01
May 3, 2024

1 Abstract

Robotics couples many aspects of Computer, Software, and Mechanical Engineering. Each of these concepts must be applied in a successful robotics platform. An investigation in the utilization of a Realtime Operating System (RTOS) was undertaken and implemented on a self-balancing robot. A RTOS was implemented as maintaining task priority is important for auxillary computation while maintaining control of a robotics platform. Potential auxillary computation is also discussed, and a metric is attained to understand the capabalities and limitations of the microcontroller unit (MCU) selected.

2 Introduction

This project consists of a MCU, acceleromter, gyroscope, magnetometer, two brushed direct current (DC) motors, and two brushed DC motor drivers. Each of these components play a pivotal role in maintaining platform attitude, communication, and computation. Tasking a robot can be a difficult task for computers that are either too simple, such as lower end arduinos, but also for platforms capable of running linux. This arises from not being able to understand a robots state often enough, but also not properly prioritizing control of a robotics platform. While control is certainly an important concept for robots, others tasks they must execute are often nontrivial. These nontrivial tasks are often related to navigation, sorting objects, or gathering information about an environment. Understanding the amount of extra computational capacity can help baseline requirements for a robotics platform to enable auxillary tasks.

2.1 The Problem

The problem of self-balancing a robot has many aspects. First, the objective is to not fall, this is best done by maintaining upright. Knowing the angle of inclintation of a robotics platform is best done using an Inertial Measurement Unit (IMU). The IMU selected for this project is a BNO-055, a microelectromechanical (MEMS) IMU. An angle of inclination is sensed at the location of the IMU. From this location it is required to calculate the respective forces to apply at the wheels. In this project we are assuming we only want to stay upright, therefore we will assume the IMU sensed pitch value as the attitude angle to minimize. This coordinate system will be known as IMU_{Body}.

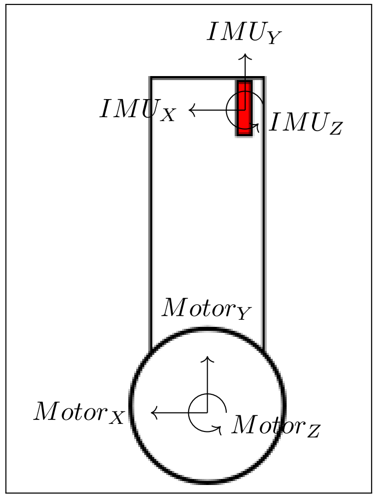

Figure 1: Coordinate Systems

Implementing a robot has multiple steps. These steps include initializing all physical states, linear and angular accelerations, velocities, and positions. From these states it is possible to derive a force required to maintain a zero angle for the robot. An optimal route for this robot would use Model Predictive Control. This particular robotics platform implementation uses a Porportional Integral Derivative (PID) controller. A PID can often be difficult to properly tune for this nonlinear system, which was what was experienced during this implementation. This robotics platform potentially suffered from flexible mechanical design, IMU noise, or an imporper Pulse Width Modulation (PWM) driver. There were many technical hurdles to address, such as implementing an I2C driver from scratch. This process proved rather difficult, references were usually improperly implemented, or did not use a modernized interface. This project brought an appreciation of modern operating systems and their driver libraries.

2.2 Literature Review

Realtime Operating Systems are essential components to systems that are highly reliant on dedicated state space control tasks. This reliance is due to the nature of systems that are highly susceptible to instablities, such as aircraft, rockets, submarines, and robots. The capability to handle multiple tasks, including dedicated control tasks is ubiquitous to many software architectures involving robotics. [1] In *Principles of guidance, navigation, and control of UAVs* the authors use multiple threads to perform navigation, sensor reading, perform research algorithms and more on a uncrewed aerial vehicle (UAV). The authors split these tasks among multiple threads using FreeRTOS, and show through flight test the ability to handle complex tasks while maintaining control of an UAV. This paper discusses a similar use case for sensor sampling as for the platform implemented in this project, and the ability to properly design noise filters.

Other works that show the capabilities of RTOS and control of robots largely focus on multitasking. This multitasking centers around being able to take multiple inputs and provide multiple outputs to a robotics system as a whole. The authors of *Embedded Implementation of a Real-time Switching Controller on a Robotic Arm* show the capacity of a robotic arm to multitask. This multitasking is implemented directly into the Operating System of the robotic arm, while maintaining absolute control of the robot, performing motion planning, and much more. The capability of this robot was all facilitated by a novel scheduler, FreeRTOS, and a STM32F4. [2]

Each paper exhibits distinct use cases for realtime operating systems. These use cases generally regard the ability to control a complex nonlinear system and perform additional tasks all from a single microcontroller. While none of these machines are particularly unique, the systems bring a unique perspective to the complexities involved with designing software with hardware. There are rarely physical systems in the modern world that do not require fine tuned controllers and the ability to perform additional tasks from a single computer. Each of the systems presented were primarily concerned with the performance of the platform implemented and focus on the abilities of RTOSs to facilitate their use cases. These use cases required fine tuned control and the ability to multitask, without multitasking these systems would not be possible to implement.

3 Methods

3.1 The Hardware

As discussed the priority of this project is to self-balance a robotics platform. To do this a certain amount of computational capability is necessary to calculate states, gather sensor input using common communication methods, and output pulse width modulated signals. The TM4C123GXL was selected to do this, it is an evaluation kit that packages a ARM® Cortex®-M4F MCU. On this MCU there is UART, SPI, I2C, and the potential to implement CAN communication to peripheral devices. The 32-bit processor can be run at 80-MHz, utilizing 256kB of flash, and 32kB of SRAM. The BNO-055 from Bosch Sensortec combines a 14-bit accelerometer, 16-bit gyroscope, a geomagnetic sensor and a 32-bit microcontroller for sensor fusion. These devices combined with a high torque DC motors with encoders provide all the necessary components to implement a self-balancing robot. [3]

There were many hardware related issues in implementing this project. Many of which come from not starting with PCB design on Day One. Prototyping often comes at the expense of risky implementations. Choosing the breadboard prototype first method of development may lead to less costs, but that did not turn out to be the case. Many microcontrollers through miswiring, leaving programs running without cooling, as well as other issues required pur-

chasing multiple MCUs. Additionally through miswiring, incorrect usage, and mechanical failure, motor drivers and sensors also failed. Through this experience future robotics platforms should prove easier to implement. [4]
[1]

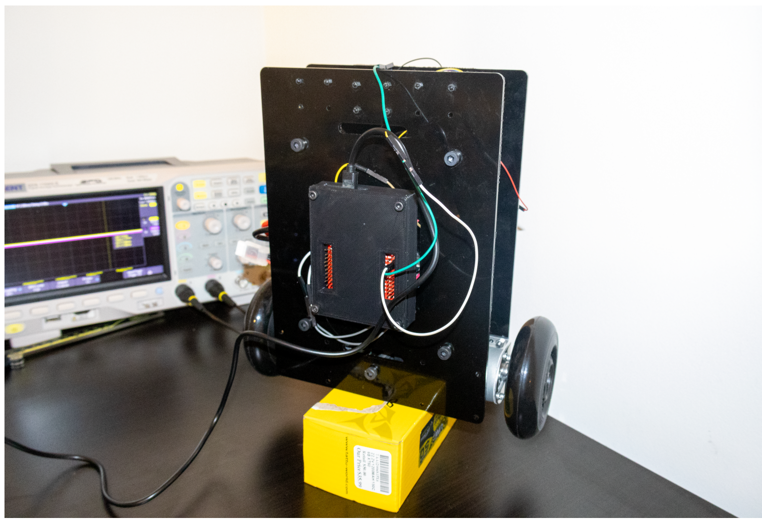

Figure 2: Roboto: Self-balancing Robot Mk.1

Component	Name	Purpose
Microcontoller	TM4C123GXL	Computation
Inertial Measurement Unit	BNO-055	Acceleration
DC Motors	Pololu Gearmotor 37D	Control
DC Motor Driver	DRV8256P	Drives the Motors

Table 1: Bill of Materials

3.2 The Software

The software consists of four major tasks, implemented using FreeRTOS on the MCU. Task One is to evaluate the state space equations, and verify that the robot is upright and send motor commands if necessary, in FreeRTOS this would be known as the 'Control' task. Task Two is to output all relevant state data for an analyst to use to understand the Robots state, and possibly other information. Task Three is to gather sensor data, the sample rate of data from an IMU can be very noisy, it is often best to sample at a rate that reduces noise as much as possible. Task Four is the 'Busy' Task. [5]

[1] There were many iterations of this robot. Four Microcontrollers have met their end in developing this hardware system.

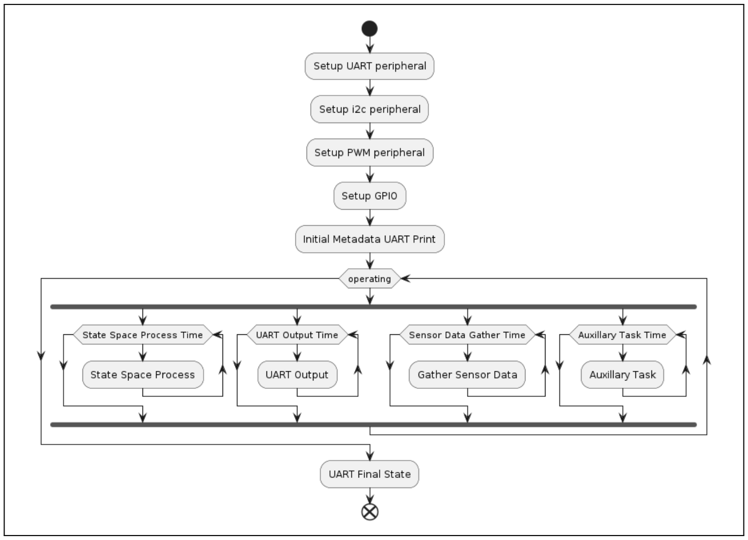

Figure 3: Robot FreeRTOS Activity Diagram

Setting up a task is fairly simple in FreeRTOS. In the listing below shows how a task is created, where the first argument is the method task to be used.

```
if(xTaskCreate(state_space_method, (const portCHAR *)"STATE_SPACE",
    STATESPACESTACKSIZE, NULL,
    tskIDLE_PRIORITY + PRIORITY_STATE_SPACE_TASK, NULL) != pdTRUE
    )
{
```

Figure 4: FreeRTOS Task Creation

With the successful creation of this task it is possible to invoke the scheduler. The FreeRTOS scheduler uses a priorty based preemption policy. Meaning, the scheduler preempts tasks based on supersceding priority, as well as if task time is available if a task is less prioritized. With this policy we have a few options, sleep or wait for a resource to become available to reengage a tasks execution. For the application of a robot, the choice was made to highly prioritize the control task. FreeRTOS uses the concept of "ticks," which are events that are scheduled every 1000 clock cycles in the current configuration of this project. With this concept it is possible to switch process execution at these events, and move onto another task if available. As it is only necessary to gather sensor data at discrete time intervals it was decided to lessen the priority of the sensor read task, and rely on short tick intervals. Additionally limitations with I2C bus clocking provide ample reason to sample the sensor at less intervals. As dynamical systems are primarily continous systems, it can be difficult to approximate continous systems on discrete systems, like computers. [6] This difficulty occurs with the inability to gather all data at all times and apply controls at all times, possibly with some latency. Accounting for this difficulty is best handled in Software, having different PID gains than you would for continous systems, implementing multiprocessing on multiple cores, and much more.

4 Analysis

The robot prooved difficult to fully stablize, although a good grasp of the interplay between FreeRTOS and Control Systems was gathered. The scheduling policy was very useful for having assured sensor data read timing, which is important when making calculations. While noise filters for this particular iteration of the robot were not implemented, they may proove useful for an eventual self-stabilizing robot. This platform served to investigate controls from a base level, using a Proportional Integral Derivative Controller, which is often only taught in simulation. Most importantly it was possible to see the impact of the update rate of the specific process controlling the robot.

During the investigation of FreeRTOS, the impact of changing the amount of time to delay the state_space_method task was shown to have system wide effects. During this investigation it was determined that updating the control task as often as possible was best, which aided in stablizing the robot more continously. The update rate was gradually reduced, which required upping the respective PID gains to have a similar effect on the robot. This effect showed that there was a correlation in control update rate and PID gains.

From the beginning of hoping to implement this project, the correlation between process time, controllability, and a computer operating systems were of primary importance. With this platform and the confidence to iterate on both the hardware and software, it should be possible to attain a stable robot, and to gather metrics on task scheduling and the interplay of control systems. Future work in this area will incorporate RTOSs on other platforms, that fly, swim, and potentially go to space. Each of these capabilities will be benefitied by knowledge gained on this project. A key metric for these capabilities, the ability to perform auxillary tasks other than control will be crucial to their success.

5 Addendum

Video of Robot Operating: https://youtu.be/vgZQlqC5u8g
Code: https://github.com/MeechaelA/roboto_freertos

References

[1] Gabriel Hugh Elkaim, Fidelis Adhika Pradipta Lie, and Demoz Gebre-Egziabher. "Principles of guidance, navigation, and control of UAVs". In: *Handbook of unmanned aerial vehicles* (2015), pp. 347–380.

[2] René Schwarz et al. "Overview of Flight Guidance, Navigation, and Control for the DLR Reusability Flight Experiment (ReFEx)". In: *8th European Conference for Aeronauticas and Space Sciences (EUCASS)*. 2019.

[3] *TM4C123GH6PM Microcontroller*. Texas Instruments. URL: https://www.ti.com/lit/ds/spms376e/spms376e.pdf?ts=1713115458962&ref_url=https%253A%252F%252Fwww.ti.com%252Ftool%252FEK-TM4C123GXL.

[4] Johan Korsnes. "Development of a Real-Time Embedded Control System for SLAM Robots". MA thesis. Norwegian University of Science and Technology, 2018.

[5] *RTOS Implementation*. freeRTOS. 2024. URL: https://www.freertos.org/implementation/main.html.

[6] Taylor P. Reynolds et al. "SOC-i: A CubeSat Demonstration of Optimization-Based Real-Time Constrained Attitude Control". In: *2021 IEEE Aerospace Conference (50100)*. 2021, pp. 1–18. DOI: 10.1109/AERO50100.2021.9438540.