Mariobots: A Lab-Based Course on Autonomous Driving Robots

A Thesis

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

---

in partial fulfillment

of the requirements for the degree

Master of Science

by

Peyman Pejman

May

2016

APPROVAL SHEET

The thesis

is submitted in partial fulfillment of the requirements

for the degree of

Master of Science

Peyman Pejman

AUTHOR

The thesis has been read and approved by the examining committee:

Joanne Bechta Dugan

Advisor

Harry C. Powell

Tom Horton

Accepted for the School of Engineering and Applied Science:

Craig H. Benson, Dean, School of Engineering and Applied Science

May

2016

TABLE OF CONTENTS

Abstract

The use of robots in the Computer Engineering department of the University of Virginia has been increasing in recent years due to their effectiveness in teaching fundamental system design and programming concepts in an entirely engaging and hands-on way. The Mariobots class takes advantage of a graphical dataflow programming paradigm to develop an autonomous robot. National Instrument's LabVIEW is the programming language of choice and myRIO is the processing unit controlling the iRobot Create. Six labs were designed to familiarize students with various Computer Engineering concepts such as signal processing, control systems, and pathfinding algorithms. The labs are tightly connected and are building blocks of an autonomous driving robot. Therefore, the labs serve as milestones toward an ultimate goal. Students begin the course by learning the fundamentals of dataflow programming. Due to the continuous nature of the labs, students are able to receive feedback on the algorithms and systems they design each session. As a result, they have the chance to improve upon their approach since almost each lab becomes an essential part of the Mariobot's behavior as the course progresses. This thesis is a compilation of the six labs that were created, including relevant background information, detailed concept explanation, and hints to possible approaches and solutions.

Preface

The content of this thesis is centered on a newly designed course, Mariobots, here at the Computer Engineering Department of the University of Virginia. The name Mariobots stems from combining the names of the robot used in the course called iRobot and the processing unit called myRIO. I have outlined the thesis to include the history and motivation behind the course. I describe various sources that inspired the creation of this course and justify the underlying educational ideas behind the creation of the course. One of the main driving factors in this course is the idea of teaching theory of various concepts in computer science or electrical engineering using practical real life lab exercises.

As the title, “Mariobots: A Lab-Based Course on Autonomous Driving Robots”, suggests the course is hands-on and involves a series of lab exercises. In addition to the concepts introduced individually in each lab the overarching motif of the course is autonomous driving. Autonomous driving of the robots is the ultimate goal and the premise of the final lab exercise in the course where all the building blocks from each lab emerge and are integrated to control the robots autonomously.

There are six labs and each one is explained in details providing background information and describing the underlying concept as well as the problem description. There are also hints, suggestions, and things to look for in good solutions to both help the instructor and students. Finally, I mention the ongoing developments and possible future additions to the course to improve the quality and enrich the learning experience.

Motivation & History

The Mariobots class evolved from a class in spring of 2014, taught by Philip Asare a computer engineering PhD student, Joanne Bechta Dugan the director of the computer engineering program, and Ronald Williams an associate professor in the computer engineering department. The class was facilitated by a teaching fellowship awarded by the SEAS (School of Engineering and Applied Sciences) graduate office. The class was titled: Model Based Engineering Embedded (MBEE) and was inspired by a class taught at University of California Berkeley. MBEE class comprised of fourth year and graduate students and the focus of the class was on developing models of systems and approaching design in a model based manner. As part of the class curriculum there were lab exercises which were adapted from the inspiration course at UC Berkeley. The platform and the equipment was however slightly different. Instead of the myRIO 1950 used in the original course, we had access to a more powerful and complete version of the processor, myRIO 1900. Additionally, Philip adopted and slightly modified a piece of LabVIEW software as a link of communication between the myRIO and the iRobot. The software was originally developed by a group called LabVIEW Hacker which has now rebranded to LabVIEW MakerHub.

I was a fourth year undergraduate student in the MBEE class and was introduced to the trio of LabVIEW, myRIO, and iRobot. The lab exercises and projects in that class encountered a number of road blocks most of which consequences of not being familiar with the programming language, the data flow paradigm, and the hardware. Once the class ended, I decided to dig deeper and learn all the nooks and crannies of the hardware and the software. Thereafter, professor Dugan decided to offer a 1.5 credit lab in fall of 2014 to continue working with the robot setup we had to explore several possibilities. I became the teaching assistant for the course and worked alongside the students to develop and design modules and state machines to serve showcase one final project. The project consisted of several robots traveling through an intersection and stopping at the traffic signal depending on the color of the light. They

stayed between lanes using a very simple strategy by employing the IR sensors. The learning curve was steep and students were lost at more points than one throughout the semester. As a result I decided to design a few check points to serve as semi lab exercises to guide the students throughout the course. That became the plan for the class offered in the following semester, spring of 2015, as a 3 credit class. The idea was to work towards a large scale obstacle course as the end goal of the class where robots avoid obstacles, read barcodes and stay within their lanes. The course was an overall success; however there was one negative consequence of my methodology: students tended to hack solutions together and spend more time optimizing parameters as opposed to programmatically and methodologically solving the problems. That experience made me realize the essential need for a series of well-structured lab exercises to introduce the concepts formally and require complete designs ever so free of tweaking parameters. As the last iteration of the class which I am part of, the students have successfully completed a series of six labs and as of the time of writing this thesis they are completing their final projects.

The remainder of this thesis is as follows: first I mention some of the related works and similar programs at other universities which help in formulating ideas for the Mariobots course, next there is a brief introduction to the course followed by a series of six lab exercises each containing introduction and objective, description of the underlying concept, statement of the problem, deliverables, and additional areas of consideration. At last there is a conclusion and future works section to layout current and possible future developments and additions to the course.

Related Works

Computer engineering, computer science, and electrical engineering concepts have long been taught with a theoretical approach throughout classrooms. However, in recent years a more hands-on approach has proven to be more effective in truly conveying the underlying concepts. Implementing an algorithm or designing a system based on a theory and then witnessing the resulting work as a physical example, greatly enhances the learning experience. Numerous engineering programs across the country are applying these methods to better serve and educate their students.

For instance, in the fall of 2010, Harvey Mudd College revamped their entire core curriculum to include more relevant elective courses. Traditionally, elective courses were solely designed for upperclassmen due to the belief that these courses had multiple prerequisites. However, they were able to design classes that would also be suitable for freshmen to allow more students the experience, without requiring extensive prior knowledge of the subject matters. [1]One of the courses Harvey Mudd College faculty introduced was titled E11: Autonomous Vehicles, which offered an interdisciplinary hands-on introduction to design and engineering concepts, ultimately motivated by a robot design competition. One of the main goals of the class was to whet the appetite of students and encourage them to seek and learn more advanced topics in the field.

[2]In a similar and related class at the Mechanical and Electrical Engineering Technology Department at Georgia Tech University, fourth year students focus on team-based, semester long projects where they design and assemble mobile robots that accomplish various tasks. This

course is particularly concerned with breaking the notion of theoretical education in engineering disciplines, specifically those concerning electrical, computer, and mechanical engineering. Most engineering programs follow methodologies regarding system design that are focused mainly on the theoretical aspects of systems. [3]This class provides students with hands-on laboratory experiences to ultimately enable more complex system designs. A significant chunk of the class is devoted to integration of embedded systems, hardware, and software.

[5]A class taught at the University of California, Berkeley as EECS 149, an undergraduate class on embedded systems, has been a role model for the creation of the course designed here at the University of Virginia. The UC Berkeley course provides students an overview to the analysis and design of computational systems that interact with physical processes. Applications of such systems are visible in just about every field, such as medical devices, traffic system control, automotive systems, energy management, consumer electronics, communication systems, etc. One of the major themes of the course is the exploration of embedded and cyber-physical systems through analysis of their interactions with the physical world. The emphasis is on basics of analysis tools, models, and design of embedded and cyber-physical systems. The physical world is modeled using continuous-time ordinary differential equations and computations are modeled using state machines. The course intends to familiarize students with formal modeling techniques, which incorporate both physical dynamics and computation. Formal techniques are used to specify, describe, and desired behavior. Lab exercises are designed to engage the students with the lowest levels of abstraction for programming embedded design systems, to the highest levels of abstractions

such as concurrent models of computation and graphical approach to system design. At the top level, LabVIEW, from National Instruments is utilized to teach model based design to students.

The robot chosen for the task is prebuilt and ready out of the box. The robot is called the iRobot Create and was first introduced back in 2007. This robot is modeled after Roomba, the vacuum cleaning automated robot, and is intended for research and development. The iRobot is particularly suitable for education due its compactness, ease of use, and robustness. [4]The iRobot has an open interface which allows it to communicate with virtually any other device or platform. The electronic interface consists of a 7 pin Mini-DIN connector and a DB-25 connector in the cargo Bay to allow other types of hardware such as additional sensors or actuators to be added. There is also a piece of software on the iRobot to allow for polling the sensors and commanding the robot. The software accepts a series of commands including mode commands, actuator commands, song commands, demo commands, and sensor commands which are sent to the robot's serial port from a computer or a micro controller connected to the Mini-DIN connector or Cargo Bay Connector.

The processing unit for computation and deployment of models to control the iRobot has been chosen to be the myRio produced by National Instruments. The myRio serves as the central controller and the link between what is developed on students PCs and what is run on the iRobot to control its' behavior and communicate with its' sensors. The National Instruments myRio is an all-purpose multiprocessor microcontroller with a Xilinx reconfigurable field-programmable gate array (FPGA). The processor is dual core that includes a fixed set of peripherals and is preconfigured with Linux including real-time extensions. The FPGA consists of

memory, logic units, general input/output, and other reconfigurable blocks at the hardware level. It is especially useful when there is a need for precise and fast parallel processing and the CPU architecture fails to deliver. A group called LabVIEW MakerHub, formerly known as LabVIEW Hacker, developed a program to be used as a platform in LabView called iRobot Navigation to deploy to the myRio as the brain of the iRobot. The iRobot Navigation project uses statechart module in LabView to allow students to easily implement their state machines. Additionally the project bundles sensor data and command information in an easy to use and compact way.

The class taught at UC Berkeley has been modified and extended to introduce a more hands-on and project focused class here at the University of Virginia. The combination of the myRio and the iRobot is called the Mariobot and is referred as such hereon. The iRobot Navigation, created at UC Berkeley, is taken as the base level of every lab and students attempt to build on top of that. The goal was to not require the students to spend much time creating interfaces between the hardware and software, but rather devote their time to designing and implementing algorithms and state machines, and see the result of their work in a real physical example using the robots. In contrast to the course created at UC Berkeley, whose primary purpose was to teach students model-based design engineering, the purpose of this course is to effectively teach students several key engineering design, computer science, and electrical engineering concepts, while simultaneously working towards creating an autonomous driving vehicle. These concepts include designing finite state machines, being able to understand and implement priority based design, learning how to design, implement and use a PID controlled system, understanding how to incorporate hierarchical state machines, designing and applying

the path finding algorithm of A*, and being able to combine all of the above, coupled with the use of a new sensing mechanism, to create a fully autonomous vehicle that can avoid obstacles and recognize traffic signals based on vision.

To fulfill this goal, six labs were created to build the course, each of which introduces a new concept that the students can use to implement a solution to the issue at hand. The resulting system designed in each lab becomes one of the building blocks of the ultimate autonomous driving vehicle. As a result, labs are continuous, connected, and improve on past implementations or designs as the labs evolve in complexity. The ultimate lab consists of autonomous driving vehicles roaming around an intricate physical recreation of roads and traffic lights.

It is important to keep in mind the cost feasibility of tools, platforms, and other needed materials when planning hands-on courses. While this thesis does not include cost analysis of the equipment used in the lab, it was necessary to mention the different pieces and their creators. The focus of this course is not to teach the students how to assemble robots or build the underlying electronics communicating with sensors. It was rather to teach more high level concepts of engineering design, computer science, and electrical engineering.

Lab01 – Stay in Bounds

Introduction

A mobile robot is an automated machine that is capable of moving through a given environment. Mobile robots have the freedom to roam around without the need of a fixed physical location, and can be further developed to be autonomous. An autonomous robot's goal is to gather necessary information from the environment it is placed in and couple that with preexisting knowledge of certain rules and regulations to achieve the utmost degree of autonomy. Such robots must work without human intervention and try to avoid situations where the safety of people or the surrounding area is compromised.

Autonomous mobile robots must possess an arsenal of sensors to extract information from their surrounding area. Such information is vital in the process of decision making. Environmental sensors come in different forms or shapes such as electromagnetic spectrum, vision, chemical, touch etc. For instance, some vacuum cleaning robots determine the amount of time they have to spend in one particular area based on the rate at which dirt is being vacuumed. Perhaps the most basic need of a mobile robot is to detect the boundaries of the environment it is traveling through. Depending on the situation and the type of robot, such boundaries could be signified with various distinctive areas such as land vs water, dirt road vs pavement, black background vs white background etc.

This lab's objective is to detect the boundaries of a road using the Mariobot's bottom infra-red sensors and to drive between those boundaries. The background color is distinctively

different from that of the boundary lanes. The Mariobot must be able to continuously drive forward and adjust itself every time it goes over any of the boundaries.

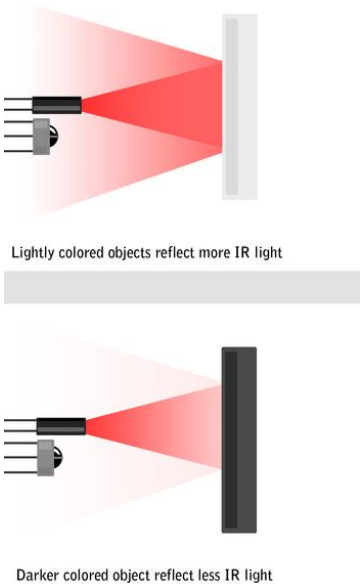
Concept

All objects emit heat energy in the form of radiation as long as they are above absolute zero temperature. Such radiation is invisible to the naked human eye, since the wavelengths are not within that of the visible light spectrum. However, such radiation can be detected through electronic sensors designed to measure infrared radiation. These sensors often act in a passive manner, meaning they do not emit any radiation for detection purposes, but rather work entirely based on radiation reflected from other objects.

The IR light entering the sensor is transformed into an electric current and thereafter detected by a voltage detector. This is due to a unique property of Light Emitting Diodes. LEDs are able to emit light at a specific wavelength when supplied a certain level of current. On the other hand, they are also able to be wired to produce a certain current when subjected to that same light radiation. IR emitters and detectors are found in many devices used every day, such as remote controls or a computer mouse. Each one uses IR for a different purpose: remote controls use it to transmit data from one device to another and a computer mouse uses it to track the direction of movement.

Infrared sensors could be used to detect the surface brightness of various objects. They are effective in distinguishing between dark and light objects. Infrared sensors, in their most basic form, come in pairs. Such pair consists of a sender and a receiver placed adjacent to one another, where the sender emits rays and the receiver records the reflection. Objects with

different colors reflect different amounts of the emitted IR. This enables us to distinguish between so called “dark or black” and “light or white” colors. The following diagram shows an example of such setup.



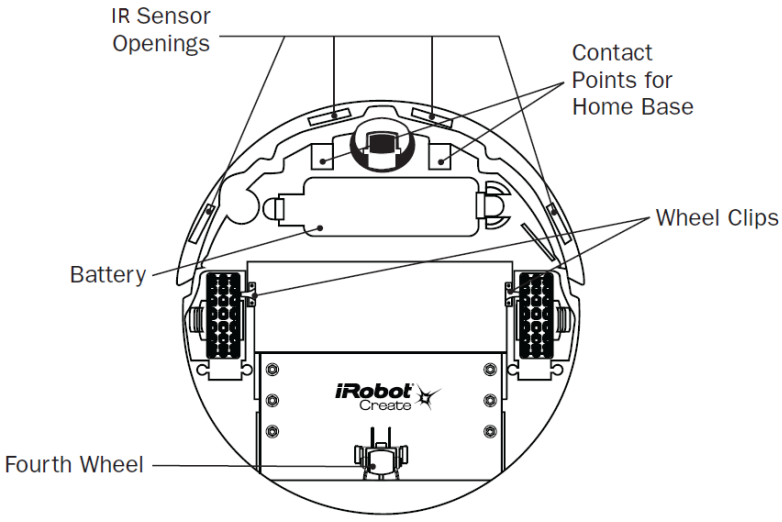
The current value detected through the IR receiver is then used to detect the brightness of object surfaces in comparison to one another. This is the underlying concept of detecting boundaries, where the background of the road is given to be vastly darker or lighter than that of the boundaries themselves.

Problem Description

In this lab we will use the IR sensors at the bottom of the Mariobot, shown in the following figure, to detect when the Mariobot has crossed the boundaries of the road. The robots will be navigating on grey mats (the road) and will have to stay within bounds of white

tapes specifying the boundaries of the road. There are various algorithms you could implement to keep the Mariobots on the road. However, keep the followings in mind when making design choices:

- Time it takes to complete the path
- Easy calibration step to adapt to the mat and tape color
- The state machine should be deterministic



Deliverables

- Detailed state machine of your design
- Demo

Additional Areas for Consideration

- This is the first step to making an autonomous Mariobot and the IR sensors are the first set of input signals. There is one set of sensors in the front and one set on the side of the Mariobot. It is important to utilize them in combination, but realize the different situations

they might be triggered in. For instance, the front sensors should be interpreted as the Mariobot is currently headed straight towards the boundary and perhaps a sharper turn is required to adjust, whereas the side ones might only require a slight adjustment. Additionally, depending on how fast the Mariobot is moving, it might be beneficiary to keep a temporary queue of sensor readings along with timestamps, to determine whether the Mariobot has fully or partially exited the track and adjust accordingly.

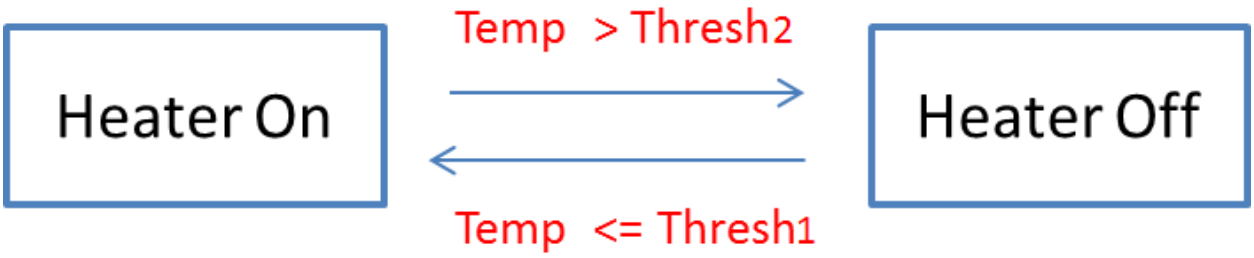
- The solution should not have the values of the white tape hard coded into the states. The value should be a control on the front panel, so that it is easily modified and adjusted for calibration purposes, or in the case, where the tape color changes.
- All four sensors do not behave the same. For any reasons including but not limited to dust accumulation, varying voltage, or simply manufacturing differences, there might be an offset associated to each one. This is resolved by correcting the value after the sensors are polled, and before they are made available to the state machine. The data cluster in the main VI of the Mariobot navigation project carrying the sensor data can be unbundled, changed, and re-bundled.

Lab02 – Reading Barcodes

Introduction

The behavior of a system or the functionality of it can be modeled in numerous ways, the oldest technique being through finite state machines. These machines describe the system by being in different states at any particular time. Each state has its own characteristics and handles inputs and outputs in a defined manner. The idea of finite state machines could potentially be traced back to some early considerations of physical matters' states. For example, water in nature is available in one of the following three states: gaseous, liquid, solid. Depending on which state it is currently in, the same chemical compound, H_2O (water), assumes different behaviors. As a result, transitions from one state to another are well-defined.

The same concept can be generalized to say that a system, whether man-made or natural, can be defined by the set of possible states that it occupies at any given time. Such states have particular behaviors associated with them, along with well-defined transitions between the states. The collection of states list, states behaviors, and transition logic comprise the definition of a finite state machine. The following figure shows a simple finite state machine with two states and two transitions. The two states describe the behavior of a heater which turns on if the temperature (Temp) drops below some threshold (Thresh1), and consequently turns off if it goes above another threshold (Thresh2). Note that this design is very simplistic and has many flaws, but nevertheless represents a finite state machine.



Finite state machines are either deterministic or non-deterministic, depending on their design. Deterministic finite state machines have exactly one transition for each input-state pair, and are additionally not allowed to make any transitions without the presence of an input. On the other hand, non-deterministic finite state machines can have zero or more transitions for each input-state transition. It is beyond the scope of this lab to provide a proof of equivalency between Deterministic Finite Automata (DFA) and Non-Deterministic Finite Automata; however, it must be mentioned that for any DFA there exists an NFA and vice-versa. Note that analysis of deterministic finite state machines is much simpler and is required for the purposes of these labs; therefore, all state machine designs must be deterministic.

The objective of this lab is to model a sequential process, such as reading a barcode, into a deterministic finite state machine, to act as the processing unit of the Mariobot reading the barcode. This lab is comprised of two phases, where phase one deals only with the solution to reading a barcode, and the second phase contains error checking, calibration, and alignment. It is important to design the state machine before implementing it, in order to identify correct transition logic, state behavior, and overall functionality.

Concept

A barcode is an optical machine-readable representation of a particular piece of data. It is much easier to recognize and understand than regular numbers . Thus, since the creation of the first barcode in the 70s, it has become a ubiquitous part of everyday life. Barcodes are found on almost every product as the standard tool of labeling. Traditional barcodes represented data by varying the widths and spacing of parallel dark and light lines. As a result, they are referred to as one dimensional or linear barcodes. The following figure shows an example of one.

The mapping between the data and the barcode is called symbology, which typically specifies two common properties. The first one pertains to the barcode being continuous or discrete, signifying the number of bars and spaces. The second one is two-width vs. many-width. In this lab the symbology is a discrete two-width, also called a binary barcode, which is defined and used to map messages or actions to barcodes. In a binary barcode there are two varying widths, “narrow” and “wide.” The width of the bars is not particularly important, and the wide bars are typically twice or three times as wide as the narrow ones.

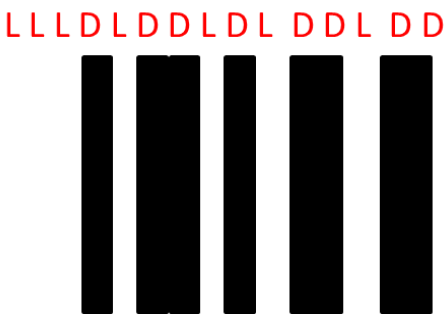
As the Mariobot moves over the barcode, it distinguishes between bars and spaces, generating a signal. This signal is essentially the input of the state machine that is in charge of

interpreting the barcode. Depending on the sequence of bars and spaces and the symbology, the state machine deciphers the message as the Mariobot goes over the barcode. Once the barcode is over, the message is complete and recorded.

Problem Description

Traditional barcodes consist of parallel lines of varying width and spacing. The barcode symbology used in this lab includes dark and light bands. The dark ones are represented with masking tape and the light ones with the background mat (road). The width of the light lines is constant throughout, but the width of the dark lines could be equal to or double the width of the light lines. As the Mariobot moves over the barcode and senses the light and dark bands, your program should detect a series of Ls (light) and Ds (dark). As a result, dark lines with twice the unit width will translate into DD, where single dark and single light lines translate to D and L, respectively.

We now introduce a pattern where a logical 0 is encoded as LD, and a logical 1 is encoded as LDD. This is where a finite state machine comes in handy to translate any given sequence in terms of L's and D's into logical 0's and 1's. The barcode is only 4 digits, which makes it easy to determine when the Mariobot should be done reading the barcode. The beginning of a barcode is specified with a single D, which means the barcode is about to begin. The finite state machine should be in a state of Wait as long as it is receiving L's (seeing the background mat). As soon as it receives the first Dark line (D), it should realize the barcode is starting. The following figure shows an example of a barcode.



As the Mariobot approaches the barcode, it is detecting Ls, which should keep the state machine in a wait state. As soon as the first D appears, the state machine should expect the barcode to start. Therefore the next L, D, and D are processed and translated into a logical 1. Then there is an L and D followed by two more LDDs, which translate into logical 0 and two logical 1s. As a result, the barcode above is equivalent to 1011.

In this lab you are given barcodes, the meaning of each code, and a set of simple actions such as speed up, turn left, turn right, play a song etc. Your Mariobot must be able to read the barcodes and use the lookup table to perform an action. The following table shows the corresponding action for each barcode.

Barcode	**Action**
1010	Turn Right
1100	Turn Left
1110	Speed Up
1111	Stop

Phase1: In this phase the direction of the Mariobot can be perpendicular to that of the barcode, such that there is no need for adjustments. Additionally, no error checking such as an incomplete barcode, a different colored tape, or a barcode longer than 4 digits is necessary. In this phase, the state machine must be able to simply identify the barcode and perform the specified action.

Phase2: In this phase your Mariobot must be able to adjust its own direction when coming to a barcode, such that it is going over the barcode in a direction perpendicular to that of the barcode lines. The barcode is placed in between lanes, as introduced in the Stay in Bounds lab, and covers the width of the road. Therefore, the Mariobot must be able to detect and stay in bounds even when reading the barcode. Additionally, incomplete barcodes must be discarded and ignored. This means that if a barcode only has 3 digits or does not follow the symbology provided, it should be ignored. It is imperative to realize the state machine before implementing, as the complexity grows tremendously in this phase and might lead to non-deterministic behavior.

Deliverables

- Detailed finite state machine of your design
- Demo

Challenge

Given the IP address of a server, establish a TCP/IP connection to communicate your barcode and receive the action corresponding to it. The communication protocol is as follows:

1. Send “H:” to establish a connection
2. If the response is also a “H:” proceed with sending the barcode by “C:XXXX” where each X is either a logical 1 or 0
3. The response will be in the form of “R:Y” where Y could be R, L, U, and S which translate to Right Turn, Left Turn, Speed Up, and Stop respectively

Additional Areas for Consideration

- Possible alignment algorithm:
    - One way to align your Mariobot with the barcode is to use the inverse tangent function in order to compute the angle at which you are hitting the tape, and adjusting accordingly. To do this, use a timer which starts as soon as either front sensor sees the barcode tape. At this point, keep track of which sensor saw the tape first, and continue driving forward. Once the opposite sensor sees the barcode tape, stop the timer and record the time. Next, compute the distance that the robot traveled by multiplying the current wheel speed with the elapsed time recorded in the last step. Then find the angle by finding the inverse tangent with the distance traveled and the distance between the two sensors on the Mariobot which is a configurable constant. Finally, drive backwards, with the wheel on the side of the first sensor moving faster until Mariobot has turned the calculated amount (in degrees). With this process the angle is effectively calculated as follows:

$$\theta = arctan\left(\frac{wheel\_speed * \Delta t}{sensor\_distance}\right)$$

- Calibration: Width of the tape should not be important since the Mariobots must use the first tape as calibration to learn the unit width
- De-Bounce: Sensors are known to bounce at the boundaries of tape and background. One solution to this is to use a de-bouncing mechanism based on time. When tape is detected first, it is validated only if it is detected some epsilon later as well.
- Priorities: it is important to realize detection priorities. For instance, it is of utmost priority to stay within bounds even if the Mariobot is in the middle of reading barcodes

Lab03 - Hill Climb

Introduction

Automatic control systems have been used in a variety of forms for over 2000 years. Early philosophers and engineers devoted their time to designing control systems with the goal of developing the idea of automation. A control system is a mechanism, device, or a process with the sole responsibility of regulating another mechanism, system, or process. Almost all ancient control systems utilized water as their controlling mechanism due to its versatility of states and availability. However, this field has matured and grown exponentially over the years and its traces are seen in every part of our modern-day lives. Most control systems have a feedback mechanism as part of their logic. The word "feedback" was formally used by radio engineers in the 1920s, referring to the parasitic, positive feeding back of the output signal from an amplifier to the input of the circuit. This 20th century word has found its place in everyday language ever since.

Robotics' and control theory's evolution have been intertwined throughout history. Early philosophy insisted on single-input single-output (SISO) linear systems, where each actuator was controlled independently. The growth in the industry has necessitated higher operation speeds of machinery, which in turn required analysis of non-linear dynamics. Such requirements motivated the development of control theory. Today, robot control systems integrate all ranges of sensors such as sonar and vision, and are highly advanced.

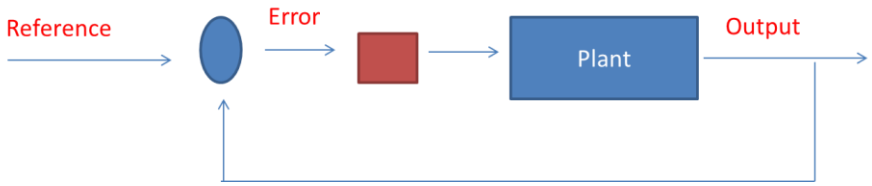
This lab's objective is to explore one of the simplest forms of a feedback system. In this lab, the robot must be able to always work towards an upward direction while navigating to the

top of an incline, without having information of its whereabouts, and regardless of any disturbances. To complete the lab, a feedback system must be implemented to control the direction of the robot.

Concept

Systems are either categorized as open loop or closed loop. The following figure is a simple example of an open loop system. In this system, the input acts on the plant or the system to be controlled, and as a result some output signal is generated. It is important to note that more often than not, the performance of such a design does not satisfy the requirements. For example, if the task is to navigate the robot from point A to point B at some distance away, one can determine the speed of the robot to find out how long the robot should drive to complete the path. This type of control is called open loop since the amount of time the robot has to drive is not adjusted based on the position of the robot at every moment. Open loop controls are sufficient for systems that do not undergo many changes and/or accuracy is not the main concern. However, in this case, any drop in voltage of the robot's battery could result in a change of speed, which is not accounted for in the calculations and thus, would result in the robot being unable to complete the path successfully.

The solution to this problem is feedback control, where the output is fed back so the system can make adjustments accordingly. In a feedback system, your ultimate goal is the target signal. This is compared to the measured value and the difference indicates the error, or in other words, how far off the system is from its goal. A controller takes in the error signal and converts it into a command that is then passed to the system as the new adjusted input. The primary goal of the controller is to minimize error over time, so that the target is reached. The following diagram shows the simple feedback system mentioned here.

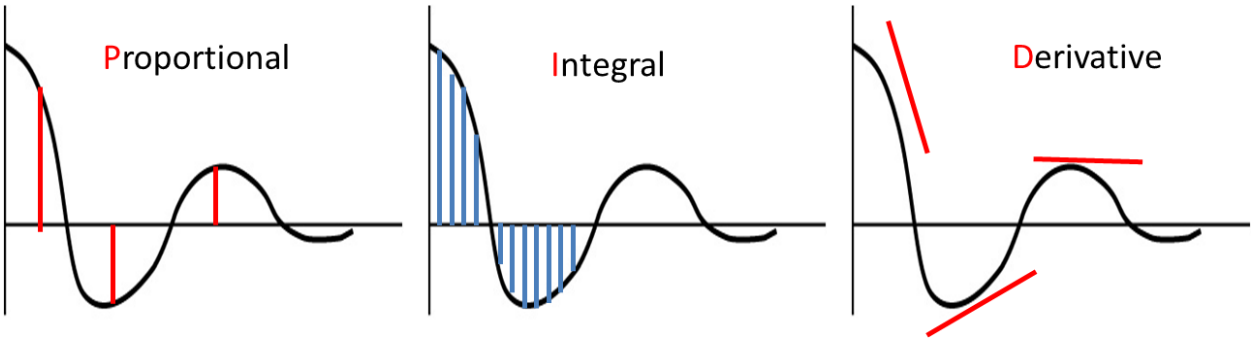


One of the most widely used control systems is called a PID (Proportional, Integral, Derivative) control loop feedback system. Each of the three terms in PID describes how the error term is treated before being summed and received by the system. In the proportional path, the error term is multiplied by a constant called proportional gain. Similarly, the integral and derivative paths are multiplied by some constant as their gains. The three terms are then summed together to make up the controller's output. The gains are adjusted or tuned and signify how sensitive the controller's output is to each of the three paths. The following equation describes the control system.

$$u(t) = K_p e(t) + K_i \int e(T)dT + K_d \frac{d}{dt} e(t)$$

$K_p, K_i \; and \; K_d$ are gain parameters

The following figure shows how each of the three components react to different error values separately. The error in the system is represented by the damped sinusoidal signal. In the proportional path, the output is the error scaled by the gain K_p. Naturally, when the error is large, the proportional path produces a large output, and when the error is negative the output is negative also. The integral path sums up the error as time goes on and multiplies it by constant K_I. It is evident that the integral path is the area under the curve. The purpose of the integral path is to remove steady-state errors, also known as constant errors, in the control system. This is useful since, no matter how small the error is, the summation would be large enough to adjust the controller's output. In the derivative path, the rate of change of the error is what is accounted for in the output signal. When the rate of change is small, the derivative path is also small, and similarly when the rate of change increases, so does the derivative path. It should be noted that not all three components are always necessary, thus their respective gains could be set to zero. For instance, if K_D is set to zero, the controller becomes of the type PI, which has only the proportional and the integral paths.



Problem Description

In this lab you will be given an inclined surface. The goal is to drive up the incline regardless of your starting position or orientation. The boundaries of the incline are marked using white tape; therefore, you should use the Stay in Bounds lab to not allow the Mariobot to fall off the edges. There will also be a few obstacles that you will need to avoid along the way. Use the implementation for the Obstacle Avoidance lab to maneuver. The obstacles will not be in a form as to corner or trap your Mariobot. They are simply placed to change the direction of motion, to test the effectiveness of your PID controller implementation. Note that your starting orientation will not be pointing down, and thus you should not be worried about finding the uphill direction.

Deliverables

- Detailed state machine of your design
- Sketch of the PID system designed with sensors and actuators labeled
- Choice of your tuning parameters and reasoning behind it ($K_p, K_i \; and \; K_d$)
- Demo

Additional Areas for Consideration

In this lab you are required to find the direction of uphill. The suggested sensor for this is the accelerometer on the myRIO. Hold the myRIO in different positions and observe the readings of the accelerometer to decide which of the axes is suitable as the input signal to your PID. You must also decide what signal it is that your controller is controlling. Once you have a

clear view of your control signal, actuators, and target point, you should be able to conquer this lab.

- Bumper sensors are far from perfect and might give off bad readings specially in head on or semi head on collisions. This results in a trap specifically when the obstacle is adjacent to the boundaries. In such cases the Mariobot should be able to use the boundaries to realize whether the direction it decided to turn to was correct and if not make proper adjustments.
- PID should work smoothly and not result in over shooting or under shooting when the Mariobot is turning. That is dependent on parameter tuning which should be accomplished manually or programmatically as a challenge using a feedback loop
- Mariobots should be able to run at various speeds to really test the feasibility of algorithms controlling them. As a result wherever possible, hard coding constants should be replaced with programmatic solutions.

Lab04 - Road Race

Introduction

Mobile robots navigate through any environment with the goal of going from one point to another, with utmost safety and a high degree of accuracy. Such navigation requires obstacle avoidance algorithms, which are slightly different than path planning. Obstacle avoidance is often designed as a reactive system, while path planning involves pre-processing and computation of a path free of obstacles to guide the robot to the target. This field has been growing exponentially in recent years due to rapid development of unmanned air vehicles and autonomous cars. There are a number of different sensors and tools that provide the necessary information about the surrounding environment for decision making. Vision, Sonar, LIDAR, and Ultrasonic sensors are just a few of such devices, often used in conjunction with one another to provide more accurate information.

The objective of this lab is to design and implement hierarchical state machines. The road race comprises of obstacle avoidance and the Stay in Bound lab. The bumper sensors on Mariobot are used to avoid obstacles. The obstacle avoidance algorithm's sophistication reduces the time it takes to complete the path. The lab is setup as a road race to encourage fast and efficient algorithms.

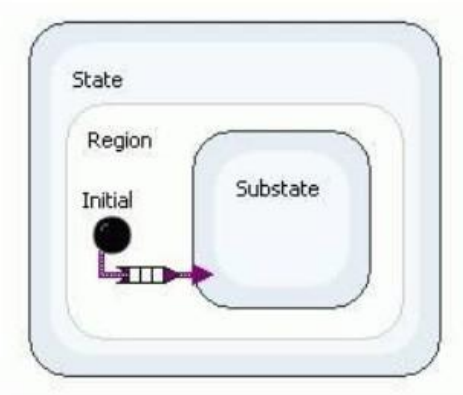
Concept

Abstraction layers help declutter and detach details from the overall vision of a solution. A simple system design of a finite state machine might include a few states, and transitions that suffice as the solution. However, a more complex system requiring a few hundred states is not

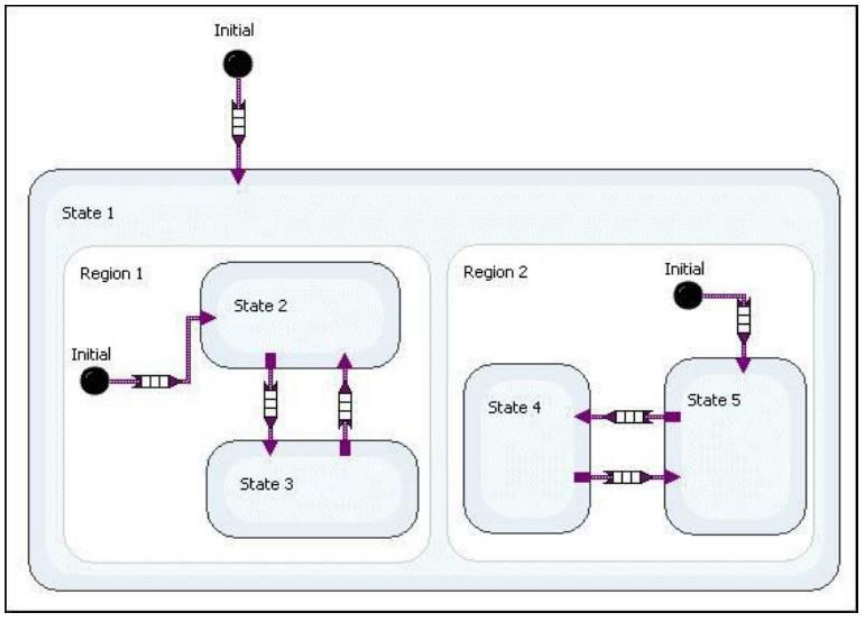
so easy to keep track of and organize when designing and implementing. One simple way to ease the organization part is the use of abstraction. Abstraction is a technique for managing the complexity of any system. It works by removing a level of complexity and establishing a new level for interaction with the system. Abstraction is a vital step to fully understanding the problem and better proposing a solution.

The idea of abstraction can very well be applied to finite state machines to ease the process and reduce the overall complexity. Theoretically, every state could represent a finite state machine in itself which determines the behavior of that state and handles the transitions in and out of it. Such a setup is called a hierarchical state machine. LabVIEW's State Char Module has the following element definitions and use cases:

- **Statechart Regions:** A region is a designated area that houses states. Hierarchical state machines are achieved through the use of regions within states. Each region needs to have an initial pseudostate for initialization purposes. The following diagram shows a simple example of a region within a state.

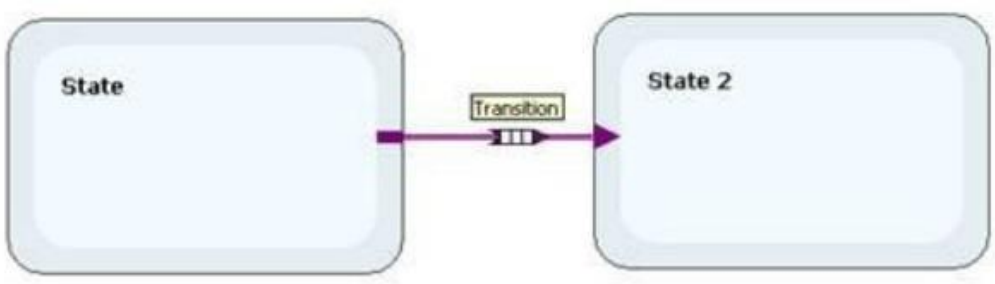


- **Statechart States:** States are placed inside regions and require at least one incoming transition. Each state has an Entry and Exit Action which execute upon entrance into the state and leaving the state, respectively.
- **Orthogonal Regions and Concurrency:** When there is more than one region in a given state, the regions are said to be orthogonal and the states within them are concurrent. For example, in the following figure, regions 1 and 2 are orthogonal and states 2 and 3 are concurrent. Furthermore, substates in orthogonal regions are also concurrent. This



means that while the superstate is active, the statechart can only be in one substate from each orthogonal region during each iteration.

- **Transitions:** Transitions define the conditions that statechart moves from one state to another. Transitions are comprised of ports and transition nodes. Ports connect the two



states together, and transition nodes specify the behavior through triggers, guards, and actions. Triggers, guards, and actions behave similar to that in states. A transition responds to a trigger and if the guard ‘s logic evaluates to true, the specified action is taken. If the logic’s result is false, then the transition is not taken and the action code is not executed.

- **Pseudostates:** a pseudostate is a statechart object representing a state including the followings:
    - **Initial State:** signifies the starting state when entering a region. Every region must have one initial state
    - **Terminal state:** signifies the last state of the region, which terminates the execution of all the other states within that region
    - **Shallow history:** signifies that when the statechart exits out of a region and returns to it, the state chart resumes by entering the highest level substates that were previously active when the exit happened
    - **Deep history:** signifies that when the statechart exits out of a region and returns to it, the state chart resumes by entering the lowest level substates that were previously active when the exit happened
- **Connectors:** a connector links multiple transition segments and comes in the following forms:
    - **Fork:** splits one transition segment into multiple segments
    - **Join:** merges multiple transitions into one segment
    - **Junction:** connects multiple transition segments

Utilizing the aforementioned tools, one can abstract away complexity, layer by layer to achieve a hierarchical design. This design technique becomes more and more relevant as the complexity and size of the finite state machine governing a system grows.

Problem Description

In this lab, the Mariobot should complete a track in the least amount of time possible. The track lane is marked with white tape, as in the Stay in Bounds lab. There are also bricks as obstacles along the track. The Mariobot must be able to avoid the obstacles by sensing through the bumper sensors and finding the right path to continue and complete the track. Each Mariobot is given a preset amount of time, in which as many trials as possible can be recorded. Each trial is timed and thus Mariobots are ranked based on their time trails. Note that obstacles are not laid out in a manner to corner Mariobots into a trap.

Deliverables

- Detailed finite state machine of your design
    - Hierarchical design is a requirement
- Demo

Additional Areas for Consideration

- Obstacle Avoidance: the algorithm used in hill climb is inadequate and must be improved. Here are a few suggestions:
    - Resume the direction of motion after circumventing the obstacle
    - When an obstacle is avoided and the Mariobot immediately sees a boundary a major correction should be done to pull out of the trap

    - Challenge: keep a temporary (time based) queue of previous actions such as turns, corrections, and backups to help make a more informed decision next
- Speed is important in this lab thus completion should not be the only goal. Speed does not have to constant through out, so it is encouraged to speed up when having the chance to. However it is imperative the barcode alignment algorithm is revisited to account for when the Mariobot runs over the tape fast enough that it passes the tape before aligning
- Hierarchical state machines are absolutely vital from this lab onward

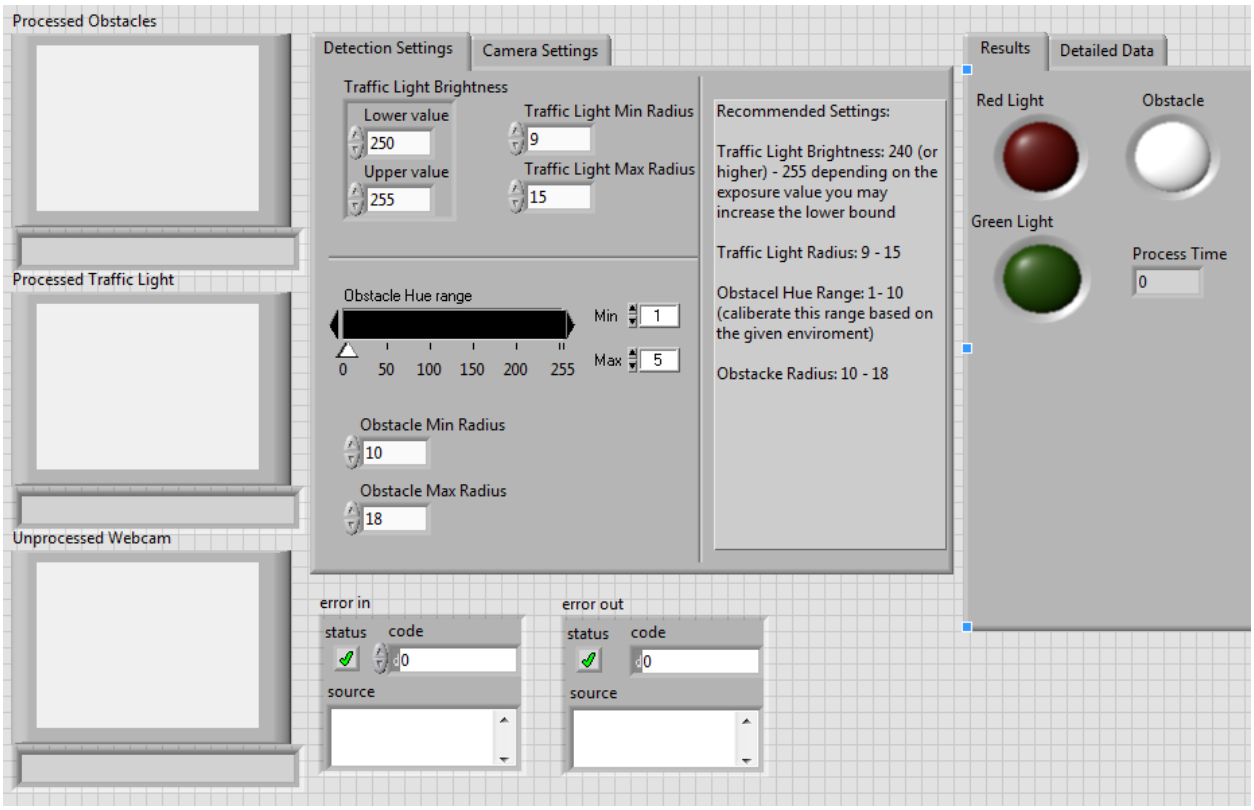
Lab05 – Autonomous Driving

Introduction

The objective of this lab is combining the pieces implemented in previous labs to demonstrate an autonomous driving robot. An autonomous robot is able to use an array of sensors to collect information about its surrounding area, in order to make safe and informed decisions while completing a task. Up until now, the robots did not have vision as part of their arsenal. Vision is introduced in this lab as an additional sensor. The vision processing unit is black-boxed and able to detect the state of traffic signals, as well as recognizing other robots based on the specific mark installed on top of them. For those interested in improving or modifying the vision unit, a detailed explanation is given in the following section.

Concept

Vision is perhaps the most vital part of an autonomous vehicle. Detecting the boundaries of the road, keeping the robot on track, detecting hazards, recognizing obstacles, and many more tasks are made possible using machine vision. For the purposes of this lab, the vision processing unit is in charge of detecting a traffic signal's state and the position of other Mariobots by identifying the mark on top of them. The front panel of this unit is displayed in the following figure and each indicator is explained below.



- Unprocessed Webcam: the real time raw feed from the camera
- Processed Traffic Light: a binary image isolating the traffic light based on intensity
- Processed Obstacle: a binary image isolating any obstacles in the image, based on color
- Detection settings: a set of parameters to tailor the detection to obtain desired results
    - Traffic Light Brightness: controls the intensity interval between 0 and 255 to isolate the light. Traffic light has a high brightness, thus the interval should be in the high range
    - Traffic Light Min Max Radius: this interval should be set depending on the distance the Mariobot should recognize the traffic light. The unit is in pixels.
    - Obstacle Hue Range: defines the interval of where the obstacle's color should fall within

    - Obstacle Light Min Max Radius: determines the minimum and maximum radius of an obstacle to be detected. This range should be chosen based on how and in what range the obstacle needs to be detected. The unit is in pixels.
- Camera Settings: depending on the quality of the camera, it may be necessary to adjust the white balance and exposure in here. The default should be set to default.
- Results
    - Red Light: Boolean indicator to show if the traffic signal is Red
    - Green Light: Boolean indicator to show if the traffic signal is Green
    - Obstacle: Boolean value to show if any obstacles are present

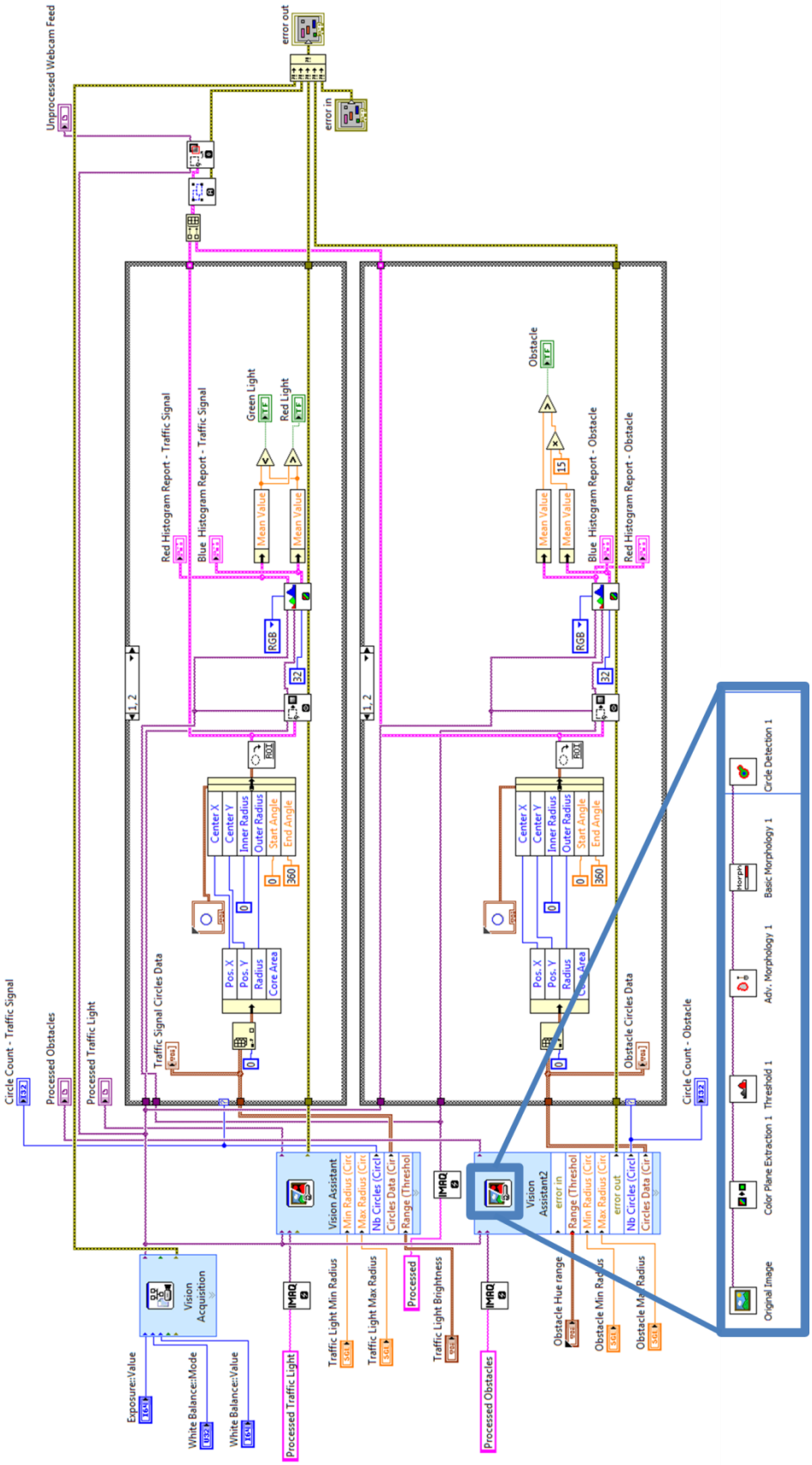
The detection algorithm for traffic signal is quite similar to that of the obstacle is outlined below:

1. Acquire an image from the webcam
2. Color plane extraction : this step extracts the specified color plane, thus changing the image into a gray scale image. HSL (Hue Saturation Luminance) is used as opposed to RGB color space to obtain color and intensity independent of one another for filtering purposes.
    a. Extract the Hue plane for obstacle detection
    b. Extract Luminance plane for traffic signal detection
3. Threshold: Use the specified ranges for obstacle's Hue and traffic signal's intensity to filter threshold the gray scale image and produce a binary image containing the desired parts

4. Morphology (Basic and Advanced): the binary contains noise and thus could cause false positives. In this stage a series of basic morphological operations such as erosion and dilation, as well as several advanced methods such as fill holes and remove small objects to clean up the image.
5. Circle Detection: circles are detected in the binary image using a Hough transform circle detection algorithm, according to the minimum and maximum specified radii. The coordinates of the circle along with its radius is recorded as output
6. Classification: the detected circles' data is processed to determine its classification
    a. Traffic signal: to detect whether the circle found in the intensity image is green or red.
        i. Based on the circle's coordinates and radius, a full color mask is created out of the original image.
        ii. Histogram of distribution of Red and Green in the mask is extracted and the mean is computed
        iii. The one with the highest mean is labeled as the condition of the color
    b. Obstacle: to detect if the circle found in the hue image is the obstacle (orange)
        i. Based on the circle's coordinates and radius, a full color mask is created out of the original image.
        ii. Histogram of distribution of Red and Blue in the mask is extracted and the mean is computed

iii. If the mean of the Red histogram is at least 1.5 times that of the blue, the obstacle is said to be detected. Such analysis could be done on the Hue plane of the HSL color space to detect other colors as well.

The following diagram shows the full block diagram of the mentioned algorithm. The vision assistant script is also shown at the bottom of the diagram in details.


Unprocessed Webcam Feed
error out
error in
Red Histogram Report - Traffic Signal
Blue Histogram Report - Traffic Signal
Green Light
Red Light
Mean Value
Obstacle
Red Histogram Report - Obstacle
Blue Histogram Report - Obstacle
RGB
32
1, 2
ROI
Center X
Center Y
Inner Radius
Outer Radius
Start Angle
End Angle
Pos. X
Pos. Y
Radius
Core Area
Traffic Signal Circles Data
Obstacle Circles Data
Circle Count - Traffic Signal
Processed Obstacles
Processed Traffic Light
Circle Count - Obstacle
Vision Assistant
Vision Assistant2
Min Radius (Circ
Max Radius (Circ
Nb Circles (Circl
Circles Data (Cir
Range (Threshol
IMAQ
Vision Acquisition
Exposure:Value
White Balance:Mode
White Balance:Value
Traffic Light Min Radius
Traffic Light Max Radius
Processed
Traffic Light Brightness
Obstacle Hue range
Obstacle Min Radius
Obstacle Max Radius
Circle Detection 1
Basic Morphology 1
Adv. Morphology 1
Threshold 1
Color Plane Extraction 1
Original Image

Problem Description

In this lab the code which identifies the state of the traffic signal through outputting two Boolean values for Red and Green light is given through a black-boxed vision unit. Additionally, the vision code is able to detect other Mariobots by recognizing a circular orange sign placed on top of each one. The information outputted is the coordinates of the center of that circle along with its radius. It is your job to decide how to interpret the information and react to it. Keep in mind that the circle will seem to increase in size as the Mariobots get closer to one another, thus you could use that information to slow down or possibly come to a full stop before crashing into other Mariobots. The boundaries and stationary obstacles (bricks) should be treated the same as previous labs. The road is marked with a single tape (same one used for barcodes) to signify the line before a traffic signal where the Mariobots must come to a stop if the light is Red.

The objective is to drive through the track while staying in your lane, avoiding obstacles, obeying traffic signals, and maintaining a safe distance with other Mariobots. You should attempt to define your algorithms and or state machines to account for all possible states. That does not mean modeling every possible event, rather it is a way of responding to certain events and grouping the rest into a default case so that your design is in a way fail safe.

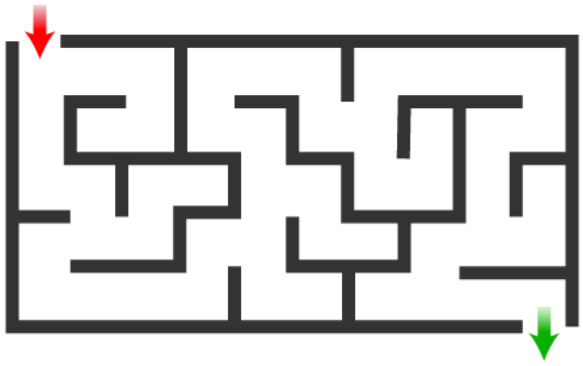
Deliverables

- Detailed finite state machine of your design
- Demo

Lab06 - Solve a Maze

Introduction

A maze is a collection of pathways or just a single pathway between two points. Mazes are built with rooms and walls, hedges, colored lines, or pretty much anything that can constrain the straight path between two points. Mazes can have one start and finish or multiple entry and exit points, depending on their designs. More complex mazes have many dead ends deep rooted in their design, which intently make them more difficult to solve. Mazes can be drawn on paper to be solved by humans, designed virtually to be solved by computers, or laid out physically to be solved by robots. Typically there are two types of maze solving algorithms. One in which the traveler has absolutely no prior knowledge of the layout, and the other where the person or the program is aware of the map beforehand and must find the shortest path. The following figure demonstrates a simple maze by highlighting the entrance and exit points.



The objective of this lab is to find the shortest path from the start to the finish point using the well-known A* search algorithm. The robot will be given a map of the entire maze with the walls and the start and finish points highlighted. The robot must then compute the

shortest path and complete the maze following that path. The mapping and position of the robot is black boxed and given using an omniscient camera overlooking the maze at all times.

Concept

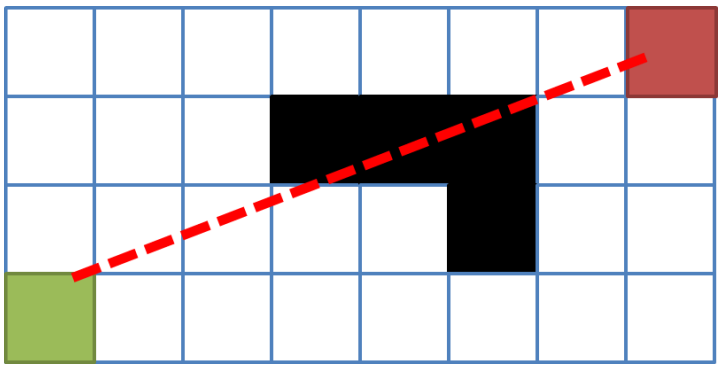
There are various algorithms that can be used to solve a maze and find the shortest path from start to finish. Dijkstra's algorithm and Bredth-first search are some examples of different algorithms that can be used. However, the algorithm that is most popular in the field of pathfinding is called A*(A-Star). A* is not a BFS (Bredth-First Search), nor it is a DFS (Depth-First Search). In fact, it is a combination of Dijkstra's algorithm and Best-First, which is a greedy algorithm. The improvement that A* makes to DFS and BFS is that it tries to pick the next node to visit not as blindly as those two do. It attempts to pick the next move based on how promising that move might be and that is where A* shines. Simply put, A* generates all the possibilities at each stage and picks the one with the least projected cost. When a possibility is generated and the associated costs are computed, they are saved in a list along with other possibilities until all of the better nodes have been searched before it.

The cost function is defined which the decision making is based on is defined in the following manner.

$$F = G + H$$

Where F is the cost of a given node and is equal to the sum of G, the cost of getting from start to that point, and H, the guess or heuristic cost of getting from that node to the target node. A heuristic is not necessarily a series of steps toward a solution; rather it helps determine the answer in an approximate form. Note that if the calculation of H is perfect, A* is capable of

finding the best path in a very short time. However, that is almost never the case and the heuristic calculation is at best, a rough estimate. The following figure shows an example map where the green square signifies the start, the red square signifies the target, and the walls are represented by black squares. In mazes such as the one below, it is feasible to choose the heuristic function to be the Euclidean distance between the node in question and the target.



Each node should have the following information stored in it: parent node, position, F, G, and H. Having a cost function for each node, the following steps should be to find the path. Two lists, one called open and one called closed can keep track of nodes as the algorithm progresses through the map. Thereafter, the following steps must be taken to find the desired path:

1. Add the starting node to the open list
2. While the open list is not empty do the following:
    a) Find the node with the least F associated with it in the open list and call it p
    b) Remove p from the open list and generate p's 8 successors and set their parent nodes to be p also
    c) For each successor check the following:
        - If the successor is the goal, the path is complete and you can thus exit

- Calculate the successor's G by adding p's G component to the distance from p to the successor. Also, the H component is naturally the distance from the target to the successor
- If there exists a node in the open list which has the same position as the successor and has a lower F, then continue and skip the current successor
- If there exists a node in the closed list which has the same position as the successor and has a lower F, then continue and skip the current successor
- if none of the conditions are met, then add the node to the open list

d) Add p to the closed list and go back to step 2

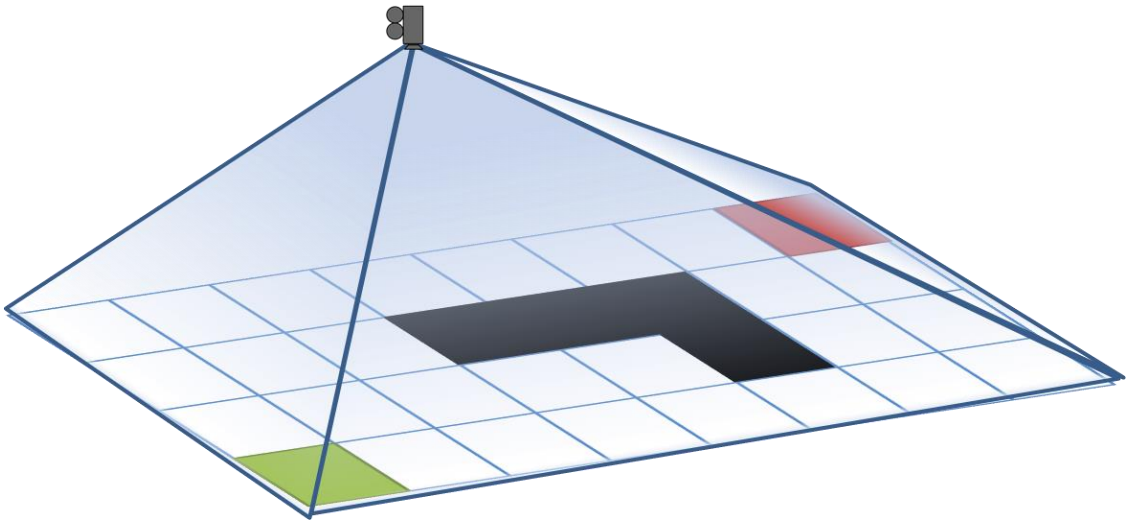
Note that when using this algorithm a node might be searched and visited many times. However, that only happens if the node is part of a better path than the last time it was part of the search.

Problem Description

The goal of this lab is to solve a maze using Mariobot. The maze is designed using squares slightly larger than the size of the Mariobot, each of which could have one of the following four different labels:

- Start (S): This square signifies the starting position of the Mariobot
- Finish (F): This square signifies the target positon
- Wall (W): This square signifies an occupied space where the Mariobot is not allowed to go over
- Empty (E): This square signifies the space that Mariobot is allowed to travel through

Details of the map are provided through a black-boxed vison processing unit that analyze the live feed overlooking the entire space with an omniscient point of view. The map is given in the form of a 2D array containing Start, Finish, Wall, and Empty as S, F, W, and E respectively, to communicate the content of each square. Once the map details are made available, your Mariobot must compute the best path from start to finish, while avoiding the walls using an implementation of the A* maze solving algorithm. Up to this point the Mariobot must remain outside of the maze. The following figure shows a sketch of the setup.



After the best path is determined, the Mariobot should be placed in the square marked as start. The vision unit provides the location of the Mariobot in a real time feed by specifying the X and Y coordinates of the Mariobot's center. Using the location of the Mariobot, the predetermined steps of the best path, and the coordinates of each square's center location, the Mariobot must navigate from start to finish. Navigation of the Mariobot is best accomplished

using a PID controller. The design is similar to that of the Hill Climb lab, with the input signal being the position of the Mariobot instead of the accelerometer's signal.

The design of this lab should be broken up into two phases- pre-process and the process itself. The pre-processing is in charge of computing the best path from start to finish and the process is to navigate the Mariobot on that path. Therefore it is highly advised to make use of abstraction tools such as sub-Vis and hierarchical state machine designs to keep everything organized, understandable, and easy to debug.

Deliverables

- Detailed state machine of your design
- Demo

Future works

There are number of different lab exercises which did not make the cut for the purposes of this thesis and have not yet been implemented. Two of the ones that should be investigated in the near future and incorporated into the course are the following:

- Calibration Lab: This lab would appear in the beginning of the series and deals almost entirely with digital signal processing (DSP) concepts. One of the major issues students deal with in various labs is the manual calibration phase of interval detection for IR sensors to differentiate between different colored tape. This lab would require the robot to drive in a straight line and record all the readings coming from the IR sensors and plot them as a time series. In addition to the background mat, the different colored tapes are laid out in parallel to one another for the robot to go over, as it continues to move in a straight line. Once the reading phase is done, the program must be able to find all the different intervals of IR readings from the colored tapes for each sensor. This lab's purpose is to thoroughly calibrate the IR sensors individually for all kinds of colored tape.
- Collision Detection Based on Accelerometer: One of the major drawbacks of detecting collisions using the bumper sensors of the iRobot is the inaccuracy and nondeterministic behavior of them. There are many corner cases where the general angle of collision is not detected properly and the robot is not able to handle the obstacle correctly. This lab attempts to take advantage of the accelerometer on board the myRIO to record the collision data. Once the data is acquired, theoretically it is possible to isolate the

collision on all major axes and determine the angle of collision using relatively simple geometry.

In addition to the aforementioned lab suggestions, there is one module that could potentially improve the ability of the robot in scanning its surrounding environment in each lab and as a result, greatly increase the degrees of freedom students have in each exercise. The module which is currently being developed takes advantage of vision to stay within the bounds of a given line using the two side lines. If and when this module is complete, it will be packaged and given to students as part of the lab files so they have yet another system to control and design various behaviors for. In short, the vision module will detect the two sides of the road and uses a PID controller to hold on to the center of the lane.

Conclusion

In recent years, there has been a strong push in engineering education to incorporate physical and hands-on projects in classrooms in order to enhance the learning experience. Traditionally, many engineering disciplines focused on analysis of concepts, phenomenon, or systems primarily using a theoretical approach. However, the creation of more project based classes has proven to not only be effective in the specific field, but it also has encouraged interdisciplinary cooperation amongst students with various backgrounds.

The Mariobot course was designed as a series of laboratory exercises that uses the aforementioned hands-on approach to familiarize students with a data flow programming paradigm, LabVIEW, as well as introducing several key engineering and design concepts in the

fields of computer science and electrical engineering. Each lab exercise introduces a concept and presents a problem statement as the objective. Additionally, each lab is a building block for a larger project, an autonomous driving vehicle, which is the ultimate goal of the class.

So far, almost three iterations of the class have been offered and each iteration has led to a series of corrections and improvements. Additionally, I have developed two vision modules to assist students with traffic signal recognition and obstacle detection. The goal of the class was to remove the heavy lifting of integrating software and hardware and dealing with low level programming assignments so that students have tools and platforms ready to use out of the box. The emphasis was on the elegant, safe, and effective design of systems to take advantage of the prepackaged software and hardware to showcase various concepts in lab exercises. The Mariobots class has gained popularity and is becoming more and more established here at Computer Engineering Department of the University of Virginia. It is worth mentioning that since the class audience is intended for third year students, the ones taking this class learn a powerful and fresh set of tools and platforms, which are used throughout the industry. The knowledge and skill set acquired are both helpful for their future careers and more importantly, broadens their horizons as they decide on the final year capstone project for their majors.

BIBLIOGRAPHY

[1] AC 2011-573: AUTONOMOUS VEHICLES: A HANDS-ON INTERDISCIPLINARY FRESHMAN COURSE

[2] An Interdisciplinary, Team-Based Mobile Robots Design Course for Engineering Technology

[3] Nielsen, M.L., “Encouraging Interest in Engineering Through Embedded System Design”, Proceedings of the 2004 ASEE Annual Conference and Exposition, Salt Lake City, Utah

[4] IRobot® Create OPEN INTERFACE. N.p.: IRobot Coporation, 2006. PDF.

[5] J.C. Jensen, E.A. Lee, and S.A. Seshia, An Introductory Lab in Embedded and Cyber-Physical Systems v.1.00, http://leeseshia.org/lab, 2013.