Group 6 Project Report

CSCI 4287

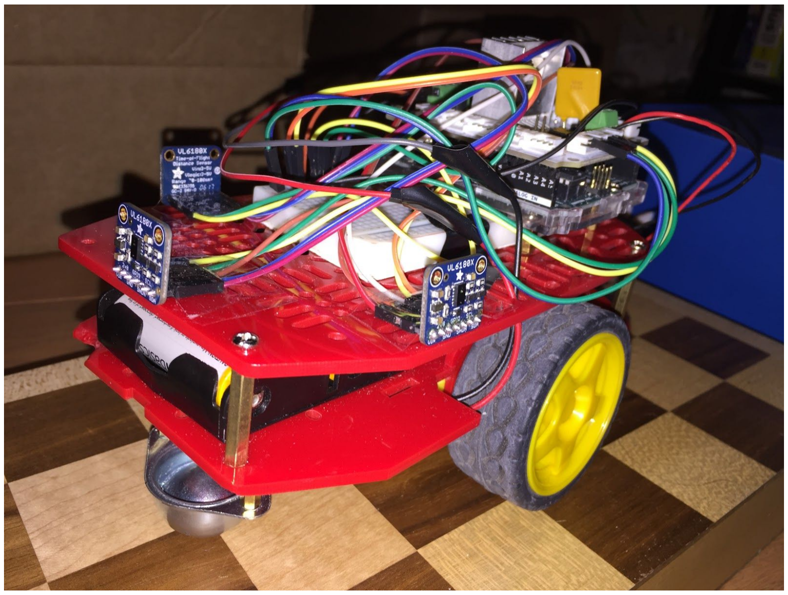
GRACE KANG
SARA KIM
TUAN PHAN
BRIAN RAWLINGS
MELANIE WOE

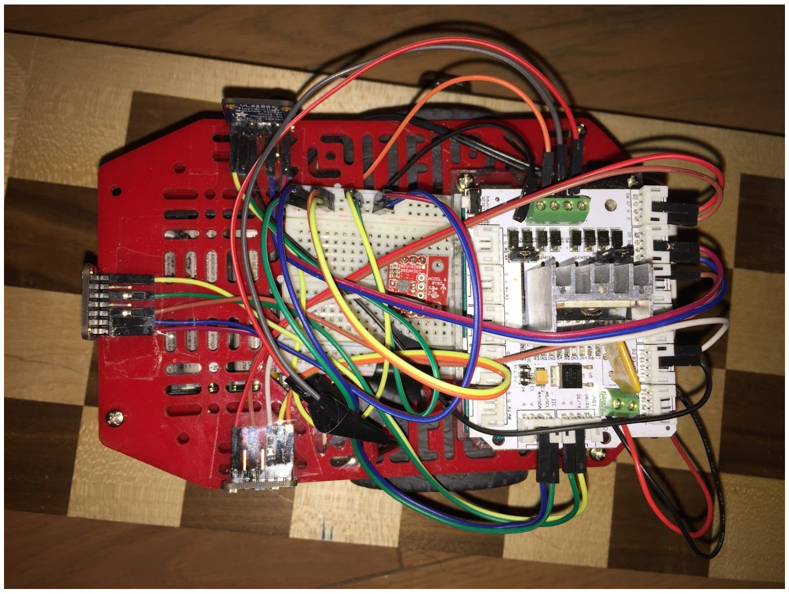
Table of Contents:

CSCI 4287

Introduction

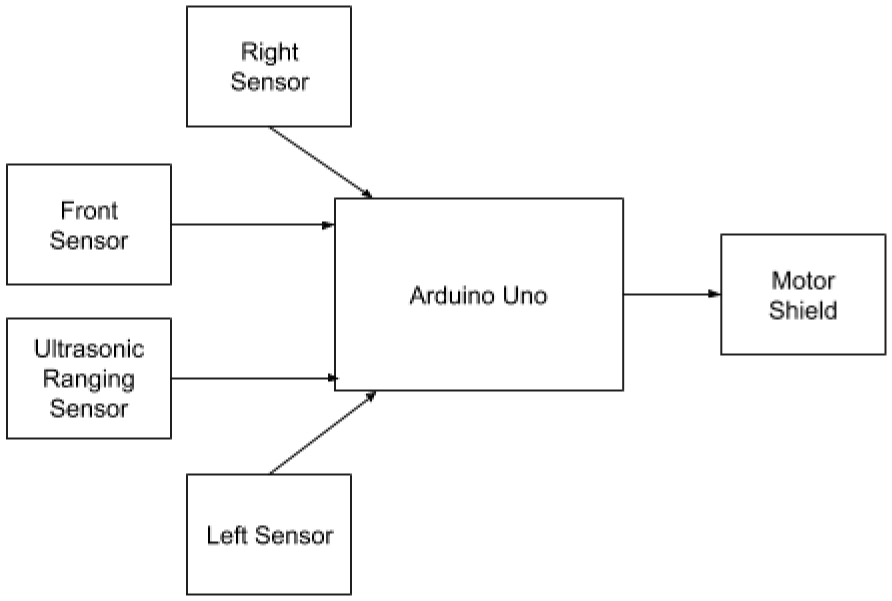
Autonomous cars are cars that can perform tasks intelligently by itself without human assistance. Provided with a ultrasonic ranging sensor, laser ranging sensor, IMU, PCDuino and its motor shield, our group built an autonomous car with an algorithm that allows it to drive automatically to a destination following a predefined path, by using input from the different sensors. For our car, if any path is blocked, the car is capable of detecting the obstacle to find another way towards the final destination. Accordingly, the car is capable of finding the exit destination regardless of where it is placed in the maze.





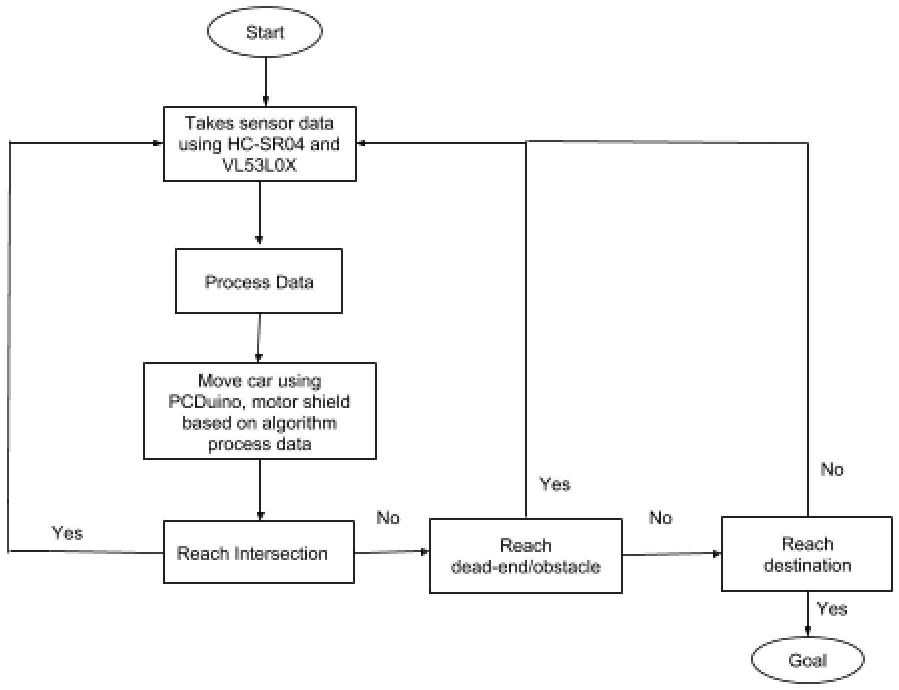
System Overview

Initial Car Design



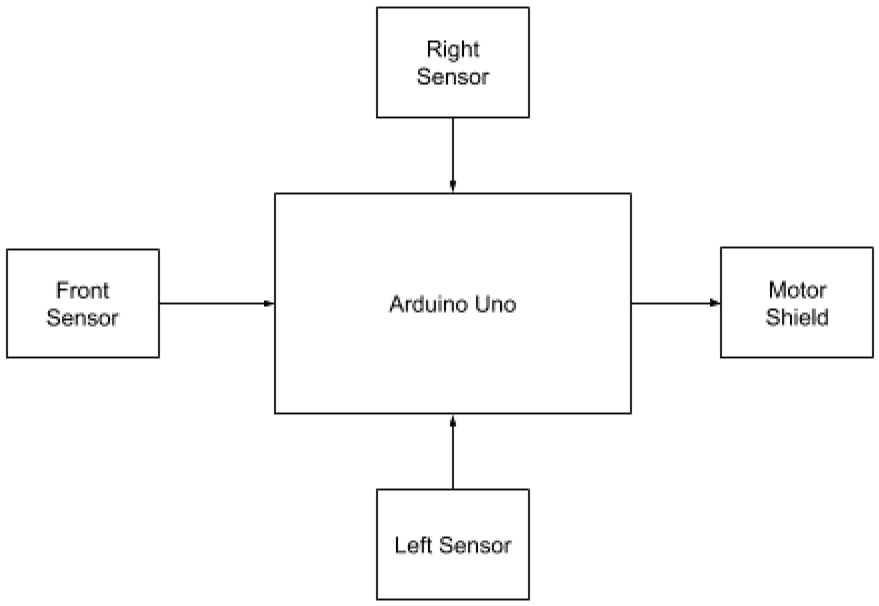
The design of the car is simple; have the 3 laser sensors point in three different directions to read any obstacles/walls in the path. The ultrasonic sensor would be used to determine how far to turn and as a back-up sensor to determine exactly how far a obstacle/wall was from the car.

Initial Control Mechanism Design


Start
Takes sensor data using HC-SR04 and VL53L0X
Process Data
Move car using PCDuino, motor shield based on algorithm process data
Reach Intersection
Yes
No
Reach dead-end/obstacle
Yes
No
Reach destination
No
Yes
Goal

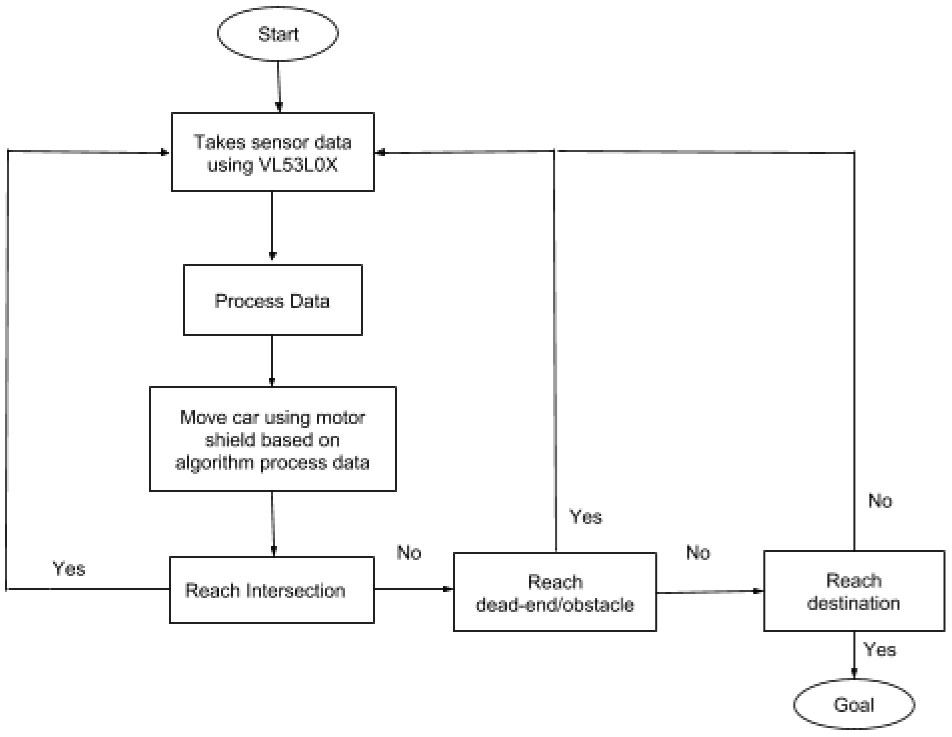
As the final design of the control mechanism did not change much from the initial design, it will be described in detail on the final control mechanism design section.

Final Car Design



The final design of the car is essentially the same as the initial design. The car has the 3 laser sensors pointing in three different directions to read any obstacles/walls in the path: 1 sensor facing the front, 1 sensor facing the right, and 1 sensor facing the left. After testing with just the laser sensors, our group decided that the ultrasonic sensor would be not be necessary and only complicate the making of the algorithm, so we removed it.

Final Control Mechanism Design



The project’s general design of the car’s control mechanism did not change much from our initial design. The design of the car system’s controlling mechanism can be divided into 4 main states: Sensor input state, process data state, direction state, and feedback state.

- Sensor Input State: The first state is responsible for reading the sensor data to later interpret and process during the next state.
- Process Data State: The process data state contains the processing of the data after the laser sensor input has been interpreted. It is during this stage that the car interprets whether there is an obstacle or wall.
- Direction State: If during the process data state the car has interpreted an obstacle or wall, the car will then begin its sequence of algorithm steps to control and move the car’s motor accordingly using the feedback state.
- Feedback State: The feedback state contains the series of checking to see if the sensor data received tells whether the car reached an intersection, dead-end/obstacle, or the final destination. Any reaching of the intersection of obstacle means that the control mechanism process starts from the beginning (take sensor input).

Challenges/Issues

- **Sensor Connections**
  The initial challenge that our team had was connecting multiple sensors together to read data. Because the laser-ranging sensors used the same address, in order to use all 3 sensors at the same time, we had to make the address all different. We created code that did just this, and we were able to use all three of the laser-ranging sensors simultaneously. However, the addresses of the laser-ranging sensors would reset to the same default address of 0x29 each time we unplugged and replugged the power to it.

  Previously, we had 2 separate programs; one for changing the addresses, and one that read in the sensor input and printed it. We had to integrate the two programs together, since we would not be able to execute two different programs during the test. Combining both codes all into one program introduced some new problems. With the previous two programs, we would initialize the sensors one-by-one: plug one in while taking the rest out, then change the address of that one sensor, while changing the code so that the next sensor that would be plugged in would have a different address than the one previously changed. This was only possible, because only one sensor was focused on at a time. If two or more were plugged in, the sensors would share the address changes. For instance, if the address of even one of the sensors changed and multiple sensors were sharing that address, it would also change the other ones. If the address was already changed to a different one, then would did not matter if the sensors were plugged in more than one at a time.

  In order to have these functions working in one program, we had to have all three sensors plugged in at the same time. Because of this, we needed to have a way to disable them. Luckily, there were freeze pins, that could freeze the pins until we decided to turn them on. Now, instead of having them all be active, we could freeze all but one sensor, change the unfrozen sensor's address, then unfreeze the others one-by-one. This solved the problem of having them share address changes, because the sensor essentially acted as if it were "removed" when frozen. With a program we found online, we were able to have all 3 sensors working and reading input in one program.

  As we got a better idea of how to get the correct input on the sensors, the cords would periodically get loose, so we would have to re-connect them. We did not find this out until later, and this was most likely the blame for some of the address changing and sensor input code issues that we had during testing, before we finally got it to work.

- **Sensor Inaccuracy**
  Occasionally we struggled with the sensors reading the wrong inputs. Our early code was written so that when the car began to approach a wall and the sensor reading goes below a certain range, the car would automatically re-adjust itself until it was center again. However, on some occasions, even when the car got too close to the wall, it would not center itself but continue heading towards the wall. Although sensor inaccuracy did not seem like a significant issue at first, because the whole car control

relies on the senor, any error in the sensors could result in the car's inability to navigate itself automatically away from any obstacles or walls.

- **Power Differences**
  We are using batteries for the motor shield and to power arduino, with that being said, the state of our car's batteries may varies. We have to adjust the car's speeds depending on the batteries' power.

- **Different Motor Speed & Inconsistency of Motor Output**
  Because of motor configurations and friction from the ground that were affecting the wheels on the motors, the car was not always able to go straight all of the time. Because of this, we had to spend some time thinking about how to overcome this obstacle. Some suggestions our team made included having the car readjust to the opposite direction when it was close to a wall (essentially having the car wiggle and center itself on the path before continuing forward), or use the change in distance of the left and right sensors at two different points in time to calculate how much to turn.

  The first solution was problematic, because of inconsistencies in the turning of the car, which will be explained later. The second solution was too complicated to implement as it required coding the timing of when to measure, calculating the turn distance, then actually turning the car. The final solution we came up with was just to go backwards far enough from the wall if it got too close and re-adjust its direction before attempting to go forward again.

  Furthermore, just by letting the wheels turn while holding the car, the right wheel turning faster than the left wheel was clearly noticeable. Even when we were testing the motors of the car with a code to simply make it move forward, the car would stray off its intended path and drive at a slight angle.

  Another problem we encountered was with turning because it was not always 100% accurate. This is problematic, because it meant that the car was not guaranteed to make it through the maze correctly.

Algorithms & Hardware

Initial Algorithm

Initially, we were considering choosing from a few different maze solver algorithms, such as the Wall Follower, the Pledge Algorithm, and Tremaux's algorithm. However, because we were running out of time with the project being harder than we anticipated and requirements being changed by the instructor due to some circumstances, we decided to at first just go with a brute-force hardcoded implementation to have the car successfully go through the maze.

Final Algorithm

We are using an algorithm that we have designed specifically for this maze. The car can start anywhere and still find its way out of the maze. Replacing the motors fixed the problem of the motors moving at different speeds, so we no longer needed a brute force approach to moving the car. With stable motor speeds, we were able to focus more on creating a better algorithm.

The final algorithm that was used works as described:

- If the car gets too close to a wall, it backs up, then turns away from the wall it was too close to.
- If the car reaches a dead-end, it backs-up, until it finds an opening. When an opening is found, the car performs a certain set of actions, depending on how many openings there are.
    - If the car is in a 4-way, it goes straight.
    - If the car is in a 3-way, it looks for an opening in this order: right, left, forward. The car will take the first open path it finds among the 3 directions.
        - For example, if the car was in a 3 way, where it had the left and forward sides open and the right side blocked, the car would take the left since the left is next in order with right not being an option (is blocked).
- If the car reaches a corner, it will take a turn.

These additions to the algorithm enabled it to complete the maze from multiple orientations and positions in the maze.

Hardware:

- Arduino Uno Board
- PCDuino
- Motor Shield
- Sparkfun IMU Breakout MPU9250
- 3 Sparkfun ToF Range Finder VL6180's
- Wires
- 2 quadruple AA External Battery Packs (One for motor shield, and another to power the arduino)
- Ultrasonic Sensor

Evaluation & Results

The car that was built and programmed for this project covers a maze solving algorithm that works specifically for this maze. We were able to collect ideal results during the testing trials by not only having the car make it through the maze, but have the car drive to the destination through different paths. Overall we found that the car is able to solve the maze correctly, even though it occasionally misses a turn at an intersection due to sensor inaccuracy. However, our group discovered that even if the car misses a specific turn that will take it to the exit, it will simply lap around the maze again to successfully find a way out (as it happened during the second test trial demonstration in class).

The only slight improvement that could be made to improve the efficiency of our car would be to have more accurate sensor readings to avoid any missing of turns, and wiring the sensors in a way that it won't disconnect easily. Our group is very satisfied with the outcome of our car because it is able to achieve the essential goal of automatically driving to a destination even when placed in various parts within the maze.

Lessons Learned

- **Process in Designing & Creating of Project**
  Working on the autonomous car has taught our group the importance and steps of a design process in an actual project. We thought it wouldn't make a difference whether or not we had a design process made. However, by using and comparing each member's process, our team was able to decide on the most fitting design and stay organized when building/coding the actual car.
- **Time Management**
  Our group struggled a bit with dividing our time to meet up together to work on the project because everyone had a different schedule. We also had difficulty trying to pace out the building and designing of the car. Fortunately, the schedule conflict actually helped our team to manage our time in an orderly manner. By scheduling different days to meet up with different members of the group, we were able to come up with a "system" to have each group continue off of what was last done by the other members.
- **Maze Solver Algorithms**
  While trying to find an adequate algorithm to use in the project, our group learned about several maze solver algorithms that can be used for the autonomous driving car. Some algorithm we considered was the wall follower, pledge algorithm, and Tremaux's algorithm. However, our team chose to instead come up with our own unique algorithm that we could flexibly use to solve the project's maze.
- **Unexpected Setbacks**
  Our group ran into a few unexpected setbacks while working on the project. Although they were minor, we were able to learn how to handle the problems efficiently even when it occured again.
  - ➔ Power Loss: The car that was working fine the day before suddenly stopped working even when it was connected to the computer or battery pack for the power source. We discovered that changing the batteries not only solved the problem, but also gave enough power to make the car move faster even with the same code being used.
  - ➔ Inconsistent Motor Speeds: While testing to see whether or not the car's motor was working correctly, our team discovered that the right wheel turned faster than the left wheel's motor.
  - ➔ Sensor Disconnections: Wires of the sensors were loose which was the reason why the sensors wouldn't activate or work sometimes. We had to make sure that all wires were inserted properly after every trial run because it disconnected so frequently.

Division of Work & Project Tasks

Grace Kang

- ➔ Design Document
- ➔ Head Organizer
- ➔ Debugger/Tester
- ➔ Secret Boss #2

Sara Kim

- ➔ Design Document
- ➔ Head of Analytics
- ➔ Head Debugger

Tuan Phan

- ➔ Design Document
- ➔ Algorithm Implementation
- ➔ Debugger/Tester
- ➔ Boss #1

Brian Rawlings

- ➔ Design Document
- ➔ Chief Mechanical Engineer
- ➔ Head Tester

Melanie Woe

- ➔ Design Document
- ➔ Director of Operation
- ➔ Head Researcher
- ➔ Best Team Leader

References

- ➢ *Learn Arduino- Getting Started*
  Arduino - Home. (n.d.). Retrieved from https://www.arduino.cc/

- ➢ *VL6180x Library for Arduino*
  https://github.com/pololu/vl6180x-arduino

- ➢ *Arduino Sketch for Multiple VL6180x Connections*
  https://github.com/luetzel/VL6180x

- ➢ *SparkFun IMU Breakout*
  https://www.sparkfun.com/products/13762

- ➢ *VL6180x Sensor Overview*
  https://learn.adafruit.com/adafruit-vl6180x-time-of-flight-micro-lidar-distance-sensor-breakout/overview

- ➢ *VL6190x: Proximity and Ambient Light Sensing (ALS) Module*
  http://www.st.com/resource/en/datasheet/vl6180x.pdf