**Team Member Responsibilities:**

Christian Alcalde:

* Proximity sensor drivers
* Obstacle detection and avoidance programs
* Power

\*Eric Fong:

* Motor control hardware and drivers
* Accelerometer drivers
* Positional feedback mapping w/ accelerometer

Abdullah Wardak:

* IR sensor drivers
* Path follower
* CPU

***\**** *= Group Leader*

**Project Name:** Autonomous Car

Our main goal is to create a car that follows a custom path made from one-inch blue painter’s tape. It will also have a method for avoiding obstacles and moving past them using a recursive function that allows the car to keep some distance from or “hug” an object until it finds the tape path again. Power will be provided to the car using four AA batteries connected in series (equivalent to 6V). Battery power will be fed into a voltage regulator that outputs 5V to the PCB VDD instead of the raw battery output. A switch on the car will power it on and off. The brushless DC motors we have listed on the bill of materials draw 750 mW each at load rating.

A simple project car chassis will be the base of the vehicle. It comes with two wheels and two DC motors, although we may use other motors if the included ones do not have good documentation or do not meet our required specifications. The car floor size dimensions are 180mm\*155mm\*3mm and the wheel size is 64mm\*64mm\*28mm. Each wheel is mounted horizontally on either side of the circular base provided. We will use washers and screws to mount our PCB to the car chassis.

Our CPU of choice will be the AT32UC3C2512C-A2UT-ND, a 32-bit AVR microcontroller that has more than enough features for our autonomous car. It requires an operating voltage of 3V to 5.5V, which will be provided through the 4 AA battery pack combined with a voltage regulator. Doing this will provide a regulated voltage of 5V. This microcontroller comes with a built-in floating-point processing unit, which will handle the necessary calculations from the IMU module. The CPU also has 45 different GPIO pins that can be used for almost anything. There is also a PWM (facilitates PWM used in motor control subsystem)module that has four different PWM channels, which is more than enough since we will only be utilizing two brushless DC motors connected to an H-Bridge. Included on the chip is a two-wire master interface (TWIM) with compatibility for the I2C standard. We will be using this communication protocol to transfer data to and from our accelerometer. A 12-bit ADC core is included with 16 channels, which we can use for our IR sensors. Connectivity through a serial interface allows for compatibility for our LCD. To program the CPU, we will be using the Joint Test Action Group (JTAG) interface. The JTAG interface is accessed through pins 1 - 4 of the CPU, through a 4-pin male header on our board. This will allow us to have four JTAG pins: TDI, TDO, TMS, and TCK.

To program the board, we will be using the Atmel-Ice Basic. On the programmer’s datasheet, the recommended header pinout is a two by five pinout for AVR JTAG. This pinout handles the four necessary pins previously mentioned, as well as GND and VCC needed to communicate with the CPU. Other pins for the programmer’s pinout are included (NC and TRST), however are not necessary for programming functionality.

Path following will be implemented using three IR reflective sensor modules, the CPU, and the motor driving subsystem. The IR sensors will observe reflective values of three spots in front of the car, one in the middle and one each to either side, and send readings to the CPU as an analog signal. The sensors will be placed on a breakout board that will be attached underneath the car towards the front bumper. A ribbon cable will then connect these sensors to a pin header on the PCB. If the CPU detects a change in readings from the sensor, it will execute one of two turning functions until the sensor output returns to the on-path values, then resume forward travel. A dark signal from the left sensor will encode for executing a left turn function, vice versa for right turn, and a dark signal in the middle with bright on the sides will encode for the on-path state. On the track there will be a wider section of blue tape that signals for the car to stop. In order for the car to stop, all three IR sensors will turn dark for a longer time. This section will extend for the length of three seconds. Once the IR sensors turn dark for three seconds, the CPU will read the signal and decide to stop.

To read data from these sensors, each one will be connected to different resistors, as shown in the recommended circuit setup found in the sensor’s data sheet. In addition to this, each sensor will need to have its own ADC to read the analog data it outputs. From here, a ribbon cable will connect to the pin header on our PCB. These pins will connect directly to the CPU because the 32-bit AVR has a built-in 12-bit ADC that has the capability to sample data from all three IR sensors almost simultaneously. We plan to have the CPU read a value from each sensor every millisecond.

The motor driving subsystem will control the speed at which the motors run and the direction they move the vehicle in. Steering will be approached by running the left and right side motors at equal speeds in opposite directions to rotate the car in place, or running one motor faster than the other. To implement this, we will have the CPU send control signals to a L293DD H-bridge IC which can change the polarity of current through each motor. These same connections can be used in conjunction with a PWM signal to vary the average voltage supplied to the motors and produce variable speed control for the motors. The L293DD can control the polarity of two motors independently.

Obstacle detection will be implemented using two ultrasonic distance sensors to detect any objects in the way of the car. The algorithm used will be explained in the next paragraphs. Each sensor will be placed with a 15 degree offset from the left and right sides of the car. This is done to avoid any ultrasonic interference from either sensor. If the CPU receives a reading from either sensor indicating an obstruction is within 100 mm of the sensor, the car will activate the obstacle circumnavigation subsystem. The distance sensors connect to the CPU using 4 pins: VCC, Trig, Echo, and GND. These sensors use a custom communication interface to send data to the CPU. Because of this, we will connect the two data pins to any available GPIO pins. The trigger pulse input pin (Trig) receives a pulse from the CPU, prompting the sensor’s transducer to send out eight 40 kHz signals. These signals will bounce on an object and back into the sensor. The echo pulse output pin (Echo) then receives this signal and performs a calculation to determine how far the object is from the sensor. The formula that the distance sensor uses for calculating the distance is the following:

Test distance = (high level time \* velocity of sound (340 m/s) )/2.

Obstacle circumnavigation will be triggered by obstacle detection, and will be implemented using the travel logging, obstacle detection, and motor driving subsystems on the CPU.

Implementation of the circumnavigation function. We will assume that obstacles occupy a footprint of no more than 60 cm x 60 cm. To determine whether to pass the obstacle on the right or left, we will compare the outputs of the distance sensors. If the left sensor reads a shorter distance than the right sensor, this indicates that there is more obstruction on the left, so the function will drive the car to the right, and vice versa for driving left. If both sensors give an equal output, the car will default to circumnavigating on the right of the obstacle.

After deciding the initial direction to circumnavigate, the car will drive forward while “hugging” the obstacle. If circumnavigating on the right of the obstacle and the left distance sensor reads a drop in distance, the car will drive its right motor slower to arc the car away from the obstacle until the left distance sensor reading matches the hugging-distance of ~215 mm. If circumnavigating on the right of the obstacle and the left distance sensor reads an increase in distance, the car will drive its left motor slower to arc the car closer to the obstacle until the left distance sensor reading matches the hugging distance of ~215 mm. The circumnavigation on the left of the obstacle is vice versa.

If the car detects that it’s back on its original path, it will exit the circumnavigation subsystem and switch back to the path following subsystem. With obstacles fitting in a 60 cm x 60 cm footprint, the car should not need to travel more than 250 cm to circumnavigate an obstacle at worst, so if it does travel more than that while circumnavigating, the car will be programmed to stop and display an error message on the LCD display. From here, the car would have to be turned off and placed back on the tape path manually.

The LCD Character Display that we are using has a display format of 16x2 characters, allowing for up to 32 custom characters to display at one time. On the display we will display various things such as the car’s acceleration, velocity, distance travelled, and error messages. To display the various statistics on the screen, we will connect a push button that cycles through the different data. The LCD has a voltage rating of 5V. This display allows for both USB and serial interfaces for communication. We will use the serial interface to avoid the need for a mini-B USB cable. There are three wires needed for the serial connection: black to ground, red to +5V, and white to the 5V TTL serial input. Once connected to the CPU via ribbon cable and pin headers on the PCB, the actual display itself will be mounted on the chassis using bolts and washers.

Our accelerometer will come from an inertial measurement unit (IMU), the SparkFun IMU Breakout board. Because this module runs from 2.4V - 3.6V, we have included an additional voltage regulator that outputs a constant 3.3V to power it.

This module is powered by the MPU-9250, a multi-chip module that serves as a 3-axis gyroscope, accelerometer, and magnetometer. However, the main use of this module will be for the accelerometer. To simplify our lives, the module has three 16-bit ADCs for digitizing the accelerometer inputs. The accelerometer will be used in conjunction with a crystal oscillator, which will be connected to the CPU, to capture more accurate data. Data will be transferred through an I2C communication protocol.

Because of the different voltages between the accelerometer and CPU (3.3V vs. 5V), we have included a level shifter circuit in our schematic. By level shifting between the two components, the IMU will be able to share its calculations in a way that the CPU understands. The heart of the level shifter is a pair of N-channel transistors, the BSS138. Each transistor allows for one level-shifting channel. When paired with several resistors, the complete level shifting circuit allows for bi-directionality, being able to go from a low to high voltage or vice versa.

Travel logging recording and display would be facilitated using the CPU, accelerometers, oscillator, 16x2 LCD character display, and some push buttons. The accelerometer will feed data over I2C connections to the CPU on the oscillator-supplied clock cycle to log the acceleration-time info of the vehicle into data structures in memory. This data will be fed into integration algorithms to obtain velocity data, which will also be fed into another integration algorithm to obtain position data. X and Y-axis data will be stored in individual data structures for acceleration, velocity, and position.

Integration will be accomplished using Riemann sums on the acceleration data to obtain velocity data, and once again on the velocity data to obtain position data. To gather the most accurate data using Riemann sums, we will implement the trapezoidal rule as it gives better results than the midpoint rule. The formula for the trapezoidal rule is the following:



Our ΔX calculation will stand for the differences of acceleration per millisecond.

In order to capture more precise data for our travel logging, we will be connecting an external crystal oscillator to the CPU. This particular crystal oscillator is a 12 MHz crystal oscillator paired with two 10 pF capacitors in parallel. According to our CPU’s datasheet, external crystal oscillators must follow certain specifications: the max frequency is 50MHz, minimum period is 20 ns, and the crystal input capacitance is 2pF. The crystal we have chosen suits our needs for precise data logging. When connected to the microcontroller, we will be able to calculate precise acceleration data from the car. The acceleration data can then be integrated to find other data such as the car’s velocity. It is also important to note that when placing the crystal on the PCB, it must be placed as close as possible to the CPU to reduce unwanted electrical noise.

An extra pin header has also been included to implement our stretch goal for log exporting to a computer. This pin header has four pins for which we will use to implement an RS232 shifter. We will use the SparkFun Electronics RS232 Shifter module, which is essentially a PRT-00449. This part allows the microcontroller to communicate with a computer. After implementing this module, we will then use a 9-pin USB 2.0 to serial cable so that we can export data from the car’s IMU module.

Travel metrics including total distance traveled, average speed, and average acceleration will be calculated from the acceleration-velocity-position data and be outputted on the display. displayed metrics will be scalar quantities calculated from the magnitudes of the x-y acceleration, velocity, and position logs. The display control and decoding will be handled by the CPU and onboard controller of the display which communicate over a parallel connection. The user will be able to cycle between which stat is displayed by pressing a button linked to the CPU. Additional buttons, LEDs, and pin headers will be connected to any remaining GPIO pins for debugging purposes.

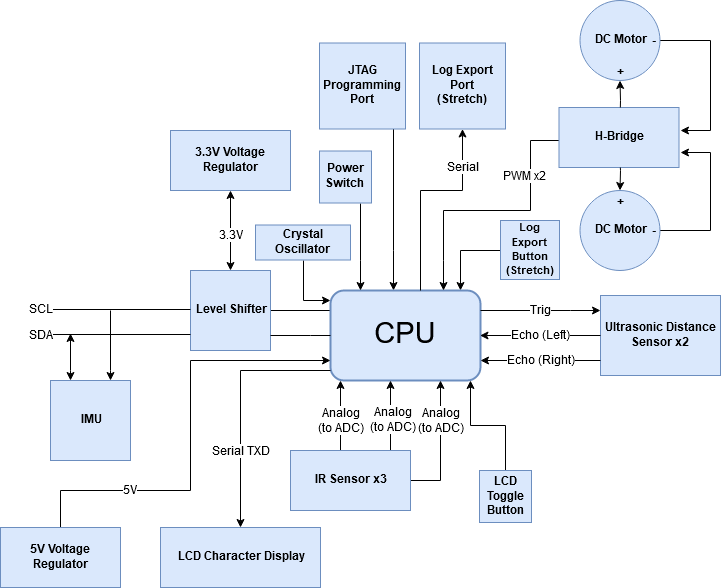
Base Objectives:

* car - controls motor steering, speed, and direction
* painter’s-tape path following
* obstacle detection
* obstacle circumnavigation
* travel statistic logging and display
  + each accelerometer can detect acceleration in three axes
  + accelerometer logs acceleration data
  + CPU integrates acceleration data to obtain velocity data
  + CPU integrates velocity data to obtain position data.
  + data for each axis is logged in its own data structure instance
  + displayed scalars calculated from magnitudes of x-y data
  + Viewable from 16x2 LED character display
    - Cycle through average acceleration, average velocity, distance traveled with the push of a button

Stretch Goals:

* advanced obstacle reaction
  + get accurate obstacle readings while car is in motion
  + detect if obstacle is moving or stationary
  + back up from approaching obstacle
  + wait out through traffic
  + add another motion sensor, detect moving obstacles via triangulation/sensor differences, avoidance swerving
* log exporting via serial UART cable
  + exporting functions triggered with a button press when plugged in
  + draw a map using x-y position data from a run and export the picture

Block Diagram



* spare CPU pins to be allocated for debugging pin headers, leds, buttons, etc. as available

PWM is a way of implementing analog voltage control using digital signaling.

Analog digital toggling voltage on and off at a high frequency given on a given period of time such that the average voltage on that period is less than digital high.

Blue toggled line represent the digital voltage and the magenta or red line is an average of Analog voltage.

IMU communicates with cpu over i2c, communicates with display uart connections.

Both USART and UART are used as a serial connectors.

Usart aren't used in our board although the cpu has the functionality.