ME 331
MECHATRONICS TERM PROJECT

AGRICULTURAL ROBOT

13.02.2021

GROUP 7

Table of Contents

Abstract

Rapid increase in global population brings global problems that require the academia, policy makers and the private sector to work together and provide sustainable, effective and profitable solutions.

Here, we present a technology demonstrator with the aim of providing a sustainable solution opportunity to the global agricultural efficiency problem by introducing a local way to the solution in our report.

This report is prepared by Bogazici University Mechanical Engineering Department students Cem Geçgel, Çağrı Ergül, Mehmet Seber and Mustafa Çağatay Sipahioğlu as the Term Project for ME331: Mechatronics Course with the help of our instructor, Assistant Professor Sinan Öncü.

1. Introduction

Climate change, one of the biggest and most important problems in our century, affects agriculture in many ways. The increase in world's temperature, changes in precipitation regimes and loss of biodiversity are only a portion of the causes for agricultural efficiency loss in farming fields.

Having access to nutritious food is the most basic and important need of life. With the rapid increase in world population, being able to provide sufficient food for the people has to be one of the main issues that governments, policy makers, unions and major groups have to address and take action for.

To provide sufficient food for people can be done in two ways. One is to provide more farming fields for agricultural use so that the overall agricultural yield is to improve, providing more food available. But this solution requires increasing the overall land used only for agricultural purposes. According to the UN's data, 10 millions of hectares of forest has been destroyed annually during the 2015-2020 period to provide more farmland to use. Losing forests for food production is not a sustainable solution for humanity, especially when preventing further CO_2 emissions and decreasing the current emissions by 2030 is becoming a part of governmental policies around the world.

One other option to produce more food is to make the production processes more efficient, thus increasing the product yield of farming lands without compromising forests. This report is focused on a project on how to improve local production efficiency with the collaboration of academia, policy makers and farmers.

Increasing the farming efficiency requires dedicated work. Since the local agricultural activities vary even in the same country, to be able to provide local and sustainable solutions is essential. A literature review has been done on the topic to identify the most logical and efficient way to produce agricultural products. When the product diversity and room to improve is considered, open field farming has the most potential to improve among other farming types.

2. Project Aim

In this project, the aim is to provide a technology demonstrator for farming fields to provide local solutions by data collection and historical data storage.

This project offers historical data storage for any farming field to provide a database to analyze and monitor the farming field as the agricultural activities go on.

3. Project Scope

For farming fields, the robot will move along predetermined points and conduct measurements in the field with regular intervals. Measurement data will be collected and synchronized to an external drive.

4. Project Management

Group 7 was formed first week, after project outline was introduced to the ME 331 class. Our group came together to do a brainstorm about possible mechatronics projects. EU Commission's and the UN's official websites were investigated to come up with an up to date project idea that acts along the goals of the EU and the UN and also can be a sustainable solution idea and technology demonstrator for Turkey's policy makers.

Our group was holding 2 meetings per week, one is for planning the upcoming week's activities and one for recording the progress and discussing the ME 331 wednesday session.

Workload was separated among 4 members of the group. While each member was focused on his own part of the project, we made sure that everyone in the group had an idea about what is being done for other processes and contributed whenever possible.

Geçgel, Cem: Embedded Control Software Development

Ergül, Çağrı: Computer Software Development

Seber, Mehmet: SDG Integration, Structural Design, Transfer Function Derivation

Sipahioğlu, Mustafa Çağatay: Electronic Systems

Team: Problem Definition, Project Ideas, Reporting

Tools Used

- Discord: Used as a communication medium for meetings and discussions.
- Google Drive: Used as a storage for all the related literature research, project processes, reports etc.
- Illustrator: Used to design a cover for the project report.
- 3DS Solidworks: Used to create a structural design for the robot.
- PROTEUS: For creating simulations of the electronics
- GitHub: For source control (Look at Appendix 14.2)
- Eclipse IDE: For writing C (The embedded software) and Java (The heatmap generator) code
- Arduino IDE: For checking and compiling C code
- MPLAB X IDE: For writing C for PICs (Not used for demo)
- WhatsApp: For small discussions
- Signal: For small discussions
- Diagrams.net: For creating flow charts
- Google Documents: For creating the report and the presentation in coordination
- Word: For creating the report
- Excel: For creating the development diary table
- Java (HotSpot JDK): For heatmap generator
- PyCharm: For writing and compiling Python (Path Finder) code
- Kite: For adding AI powered code completions to code editor
- Python: Used libraries:
    - Py2Exe: For using path finder program in Windows without needing Python
    - Tkinter: For being able to set up the User Interface
    - Struct: For having structs like C
- MikroC PRO for PIC: for PIC coding.
- WinImage: for creating a virtual SD Card image for the Proteus simulation.
- MiniTool Partition Wizard: SD Memory Card Formatter for formatting the physical SD Card.
- Fritzing,: for the Arduino schematic.

5. Sustainable Development and SDG Integration

What is “Sustainable” Development?

Throughout history, the population around the globe has grown as we humans, have developed science, culture, economy, production and industries. Unfortunately, the baselines of these pillars of humanity did not take into account that our planet, the Earth, has a limited amount of resources and we have a responsibility to provide a healthy and accessible environment for future generations to meet their own needs.

Sustainable development is a concept defined as “The development that meets the needs of present without compromising the ability of future generations to meet their own needs.” [5.1] by the United Nations. Sustainable development aims to improve the wellbeing of everyone, everywhere. In order to achieve that goal, the UN has set 17 goals back in 2015 to serve as frameworks for all kinds of groups that work for a better future.

Member states of the UN, several society stakeholders and major groups have an agreement that these goals will be integrated into the projects, production value chains, policies etc. to provide a more sustainable world for future generations.

The European Commission presented the “European Green Deal” in 2019 with the objective to be the first climate-neutral continent by 2050. The European Green Deal has roadmaps, action plans and frameworks focusing on various topics to achieve a clean, circular economy, restore biodiversity and cut pollution.

The agricultural applications are being mentioned in the Green Deal under different topics since accessibility to sufficient nutrients is one of the basic and most important needs of life. With the expected increase in world population, providing nutritious food for everyone and doing that in a sustainable way is a hot topic among the world of science. There has to be sufficient food source for the increasing population and the simplest solution to this problem is to have more agricultural fields to farm. But to provide nutritious soil for farming, lots of woodland gets destroyed. Instead, the farming fields can be used in a more efficient way with the collaboration of technology, science and agriculture.

Our project aims to be a part of this multidisciplinary collaboration and serve as a technology and solution demonstrator for research groups. There are 5 different Sustainable Development Goals integrated into our project along with the processes and results.

5.1 Sustainable Development Goal #2

"End hunger, achieve food security and improved nutrition and promote sustainable agriculture."

Eliminating hunger and promoting sustainable agriculture has to be one of the most important hot topics in science. According to the UN data, almost 690 million people were undernourished in 2019, up by 60 million from 2014. This indicates that one of the most basic needs of people, that being availability of food, is not being met for nearly 10% of the world's population.

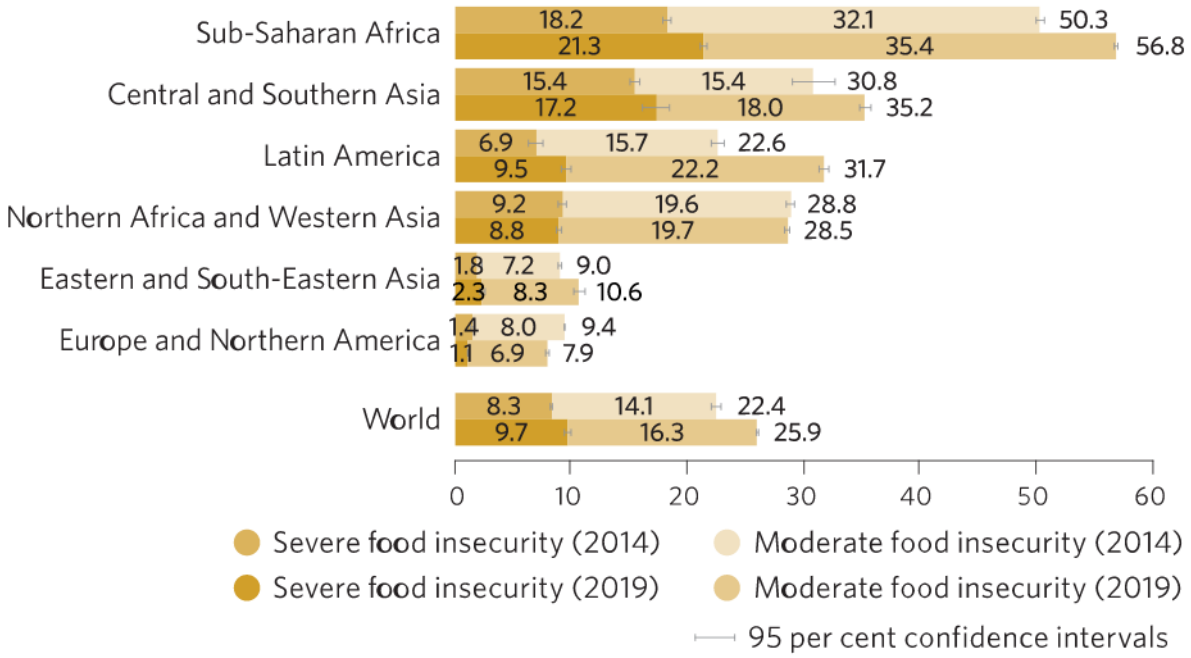


Figure shows the prevalence of moderate and severe food insecurity averages for 2014 and 2019. Around the globe, prevalence of severe food insecurity has increased by 1,4% and prevalence of moderate food insecurity has increased by 2,2% from 2014 to 2019. This indicates that food insecurity is long past the point that it has to be treated as a major crisis, Especially in Africa, Asia and Latin America regions.

In our project topic selection, we wanted to touch upon a major global problem that is getting more and more important due to the COVID-19 pandemic affecting global food systems. Our project serves the SDG #2 by promoting accurate and historical data collection and storage for farming fields and providing a database for local farming hotspots. These databases can be used by policy makers to improve the overall farming efficiency and provide job opportunities in rural places.

5.2 Sustainable Development Goal #6

"Ensure availability and sustainable management of water and sanitation for all."

During irrigation applications in farming, due to not harm the plants and the soil, clean water is being used. Also, use of groundwater collected from wells and clean water from water supply networks is being used in rural places. Eliminating excess groundwater use from wells and managing clean water use for irrigation purposes is very important to prevent water scarcity levels to rise and promote sustainable growth in our country.

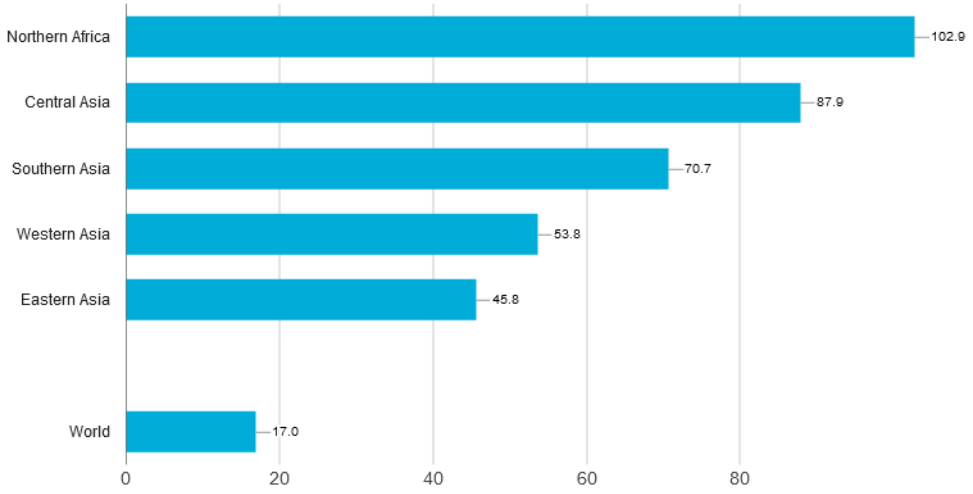


Figure shows the water stress levels (freshwater withdrawal as a proportion to available freshwater sources) for sub-regions and around the world in 2017. Industrial and agricultural water use is one of the leading reasons of water scarcity. The UN states that "Increasing agricultural water productivity is a key intervention for improving water-use efficiency".

In our project, by providing accurate and historical humidity levels for the soil, one of the side goals is to improve irrigation efficiency and decrease agricultural water use in farming fields. The UN also provides funding to projects that has an objective to provide local or global solutions related to SDG #6.

5.3 Sustainable Development Goal #12

"Ensure sustainable consumption and production patterns."

According to the UN's data, the global material footprint has increased up to 85,9 billion tons in 2017 from 73,2 billion tons in 2010. This indicates that the world is continuing to use natural resources in an unsustainable and irresponsible way. One of the opportunities that the COVID-19 pandemic offers to our society is to be able to develop sustainable recovery plans that serve as tools to build a more sustainable future in a smooth way.

From 2017 to 2019, 79 countries and the EU reported at least one policy to promote sustainable consumption and production. Recently, the European Green Deal proposed by the European Commission provides an action plan to reach a sustainable and responsible future.

Figure shows the proportion of food lost after harvesting and during transport in 2016 around the world. Our project aims to serve the SDG #12 by promoting improved and more efficient agricultural systems and local databases, sustainable agricultural production. By improving the harvesting and farming processes to be more responsible and sustainable, the food losses can be decreased in these processes.

5.4 Sustainable Development Goal #15

"Protect, restore and promote sustainable use of terrestrial ecosystems, sustainably manage forests, combat desertification, and halt and reverse land degradation and halt biodiversity loss."

Conservation of terrestrial, freshwater and mountain ecosystems has to be one of the main concerns while managing the land use. According to the UN, around two billion hectares of land on earth has already degraded, affecting around 3 billion people, driving species to extinction and intensifying climate change. On the other hand, only a third of 113 countries were on track to achieve their national targets to integrate biodiversity into their national planning policies. During the period of 2015 to 2020, 10 million hectares of forest has been destroyed annually, mainly by agricultural expansion.

Increasing biodiversity loss, combined with deforestation and land use change causes the wildlife to search suitable areas that provide shelter and food. This increases the possibility of wildlife to get in touch with civilizations more often and forms pathways for infectious diseases, like COVID-19, that threatens global health to transmit between species.

Our project aims to decrease deforestation caused by agricultural expansion by promoting sustainable solutions for efficient agricultural processes and increasing farming efficiency and decreasing the land use needed for sufficient yield.

5.5 Sustainable Development Goal #17

"Strengthen the means of implementation and revitalize the global partnership for sustainable development."

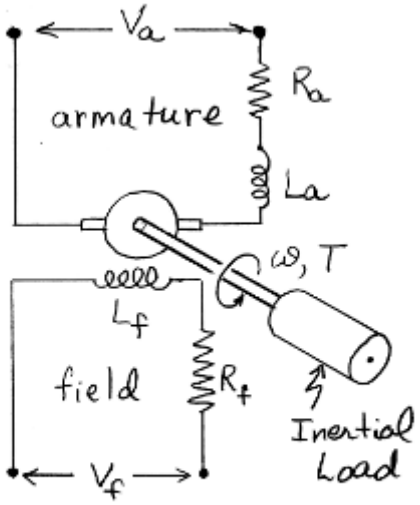
According to the UN data, global foreign direct investment is expected to decline up to 40% in 2020 and after COVID-19, remittances to low and middle income countries, which is an economic lifeline for many poor households, are projected to fall from $554 billion in 2019 to $445 billion in 2020.

Figure shows that the trend of the share of goods in exports has potential to grow in both developing and least developed countries.

Our project serves the SDG #17 by increasing the agricultural yield of local farming fields and providing more products to be exported or used within the country. Also by promoting a sustainable, rational and profitable application, our project provides a reasonable funding and business collaboration opportunities for developed countries.

6. Transfer Function Derivation

A general circuit diagram for a DC Motor is used as a model for transfer function derivation.



DC Motor

Torque generated is proportional to i_f and i_a.

$$T_M = K * i_f * i_a$$

In this project, armature-current controlled motors were used as actuators.

For an armature-current controlled motor, i_f is held constant and the armature current is controlled through the armature voltage V_a, then;

$$T_M = K_{ma} * i_a \tag{6.1}$$

Laplace Transform of 6.1;

$$\frac{T_m(s)}{I_a(s)} = K_{ma} \tag{6.2}$$

Voltage relation;

$$V_a = V_R + V_L + V_b \tag{6.3}$$

Where;

V_b: Back EMF, proportional to the speed

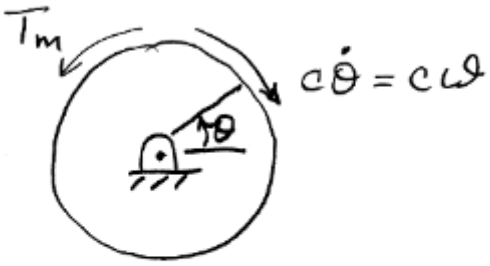
$$V_b(s) = K_b \omega(s)$$

Laplace Transform of 6.3;

$$V_a(s) - V_b(s) = (R_a + L_a s) \tag{6.4}$$

$$V_a(s) - K_b \omega(s) = (R_a + L_a s) \tag{6.5}$$

Rotation of the Motor



Free Body Diagram of the Inertial Load

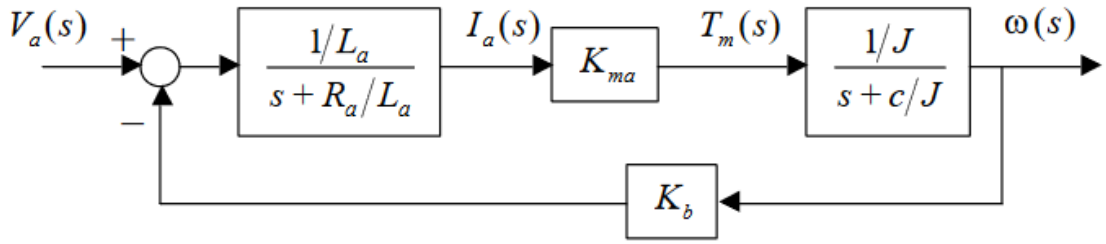
$\sum \quad M = T_M - c\omega = J\frac{d\omega}{dt}$ (CCW Positive)

$$J\frac{d\omega}{dt} + c\omega = T_M \qquad (6.6)$$

Laplace Transform of 6.6;

$$\frac{\omega(s)}{T_M(s)} = \frac{(\frac{1}{J})}{s+(\frac{c}{J})} \qquad (6.7)$$

Equations 6.2, 6.5 and 6.7 can be represented by the closed loop block diagram;



Transfer function derived from block diagram that gives resulting rotational velocity relative to the input armature voltage is;

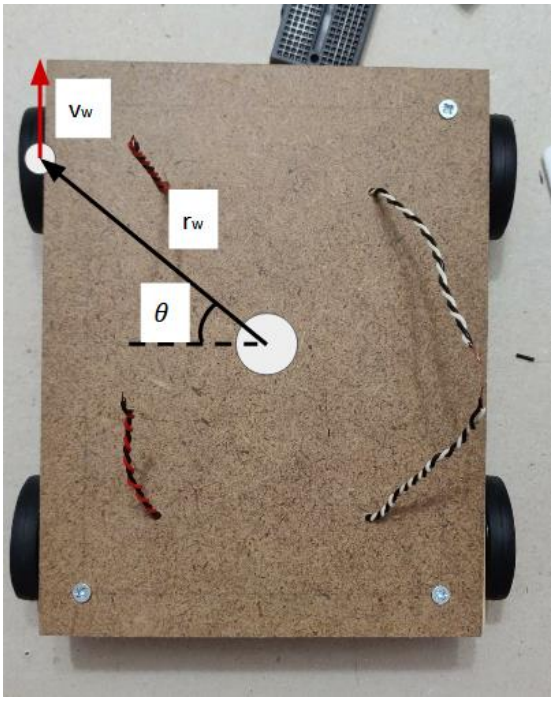
$$\frac{\omega(s)}{V_a(s)} = \frac{K_{ma}/L_aJ}{(s+R_a/L_a)(s+c/J)+(K_bK_{ma}/L_aJ)} \qquad (6.8)$$

Where;

K_{ma}: Motor torque constant for an armature-current controlled motor

K_b: Back EMF constant

Rotation of the Body



Rotation rate of the body can be expressed as;

$$\omega_b = \underline{v_w} \times \underline{r_w} \tag{6.9}$$

Where;

ω_b: Rotation rate of the body

v_w: Tangential velocity of a wheel

r_w: Distance between the centers of the body and a wheel

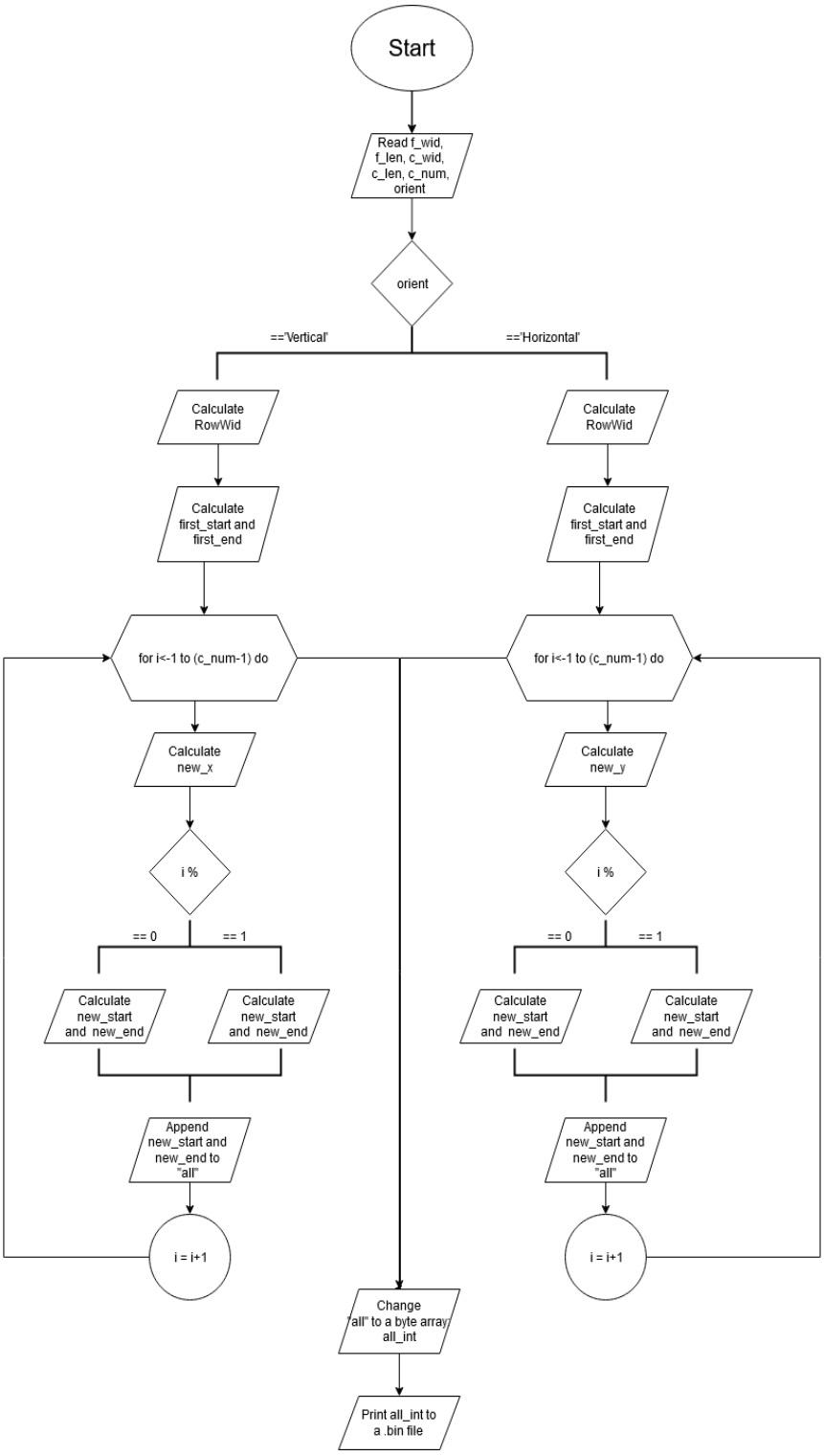
The value of tangential velocity of a wheel is proportional to the radius of the tire and output rotational velocity of the motor;

$$v_w = \omega * r_t \tag{6.10}$$

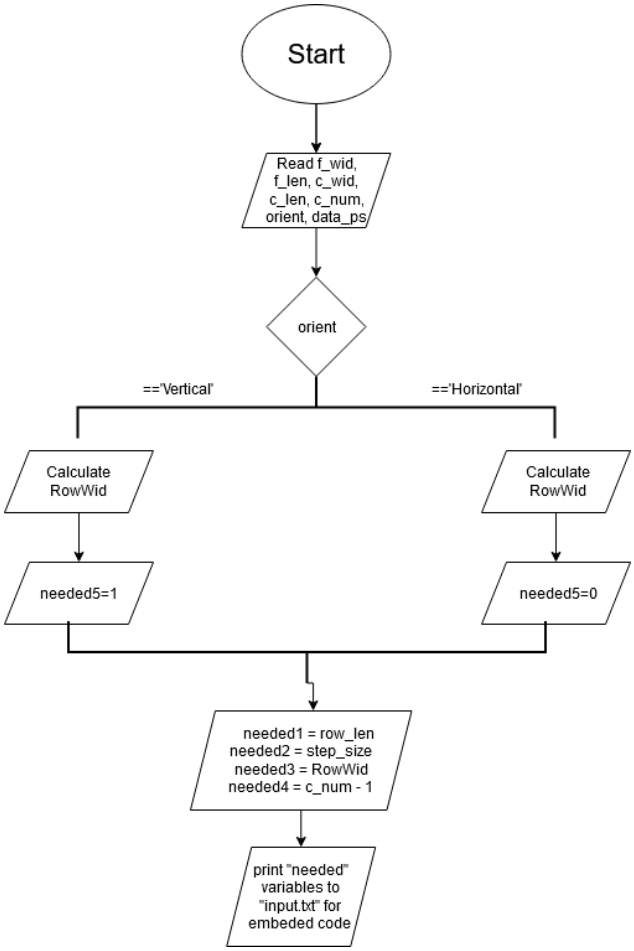
7. Computer Software Development

Path Finder

Path Finder (Actual) Algorithm Flowchart



Path Finder (Demo) Algorithm Flowchart



Path Finder Development Process

The development of this algorithm changed a lot during the course of the project.

At first, the project's scope included every shape of field. This includes but is not limited to convex and concave that have non-parallel sides. This was causing too many issues to handle. However, with the advice of the instructor, we decided to focus on only rectangular fields.

Actual Path Finder, finds every start and finishing point of the path that the device needs to pass. That way, the device only needed to go from one point to the other. For that reason, it can be implemented with little to none improvement to the actual device if necessary

However; after we simplify the field features, taking every point becomes redundant. After calculating the first start and finishing points, it becomes an iterative task, which can be done efficiently within the device itself. Otherwise, it would use so much memory storage and processing power.

For that reason, we decided to implement only the basics of the field features needed to transfered to the device, which shortened the path finder algorithm noticeable amount.

Temperature Mapping

by: Cem GEÇGEL

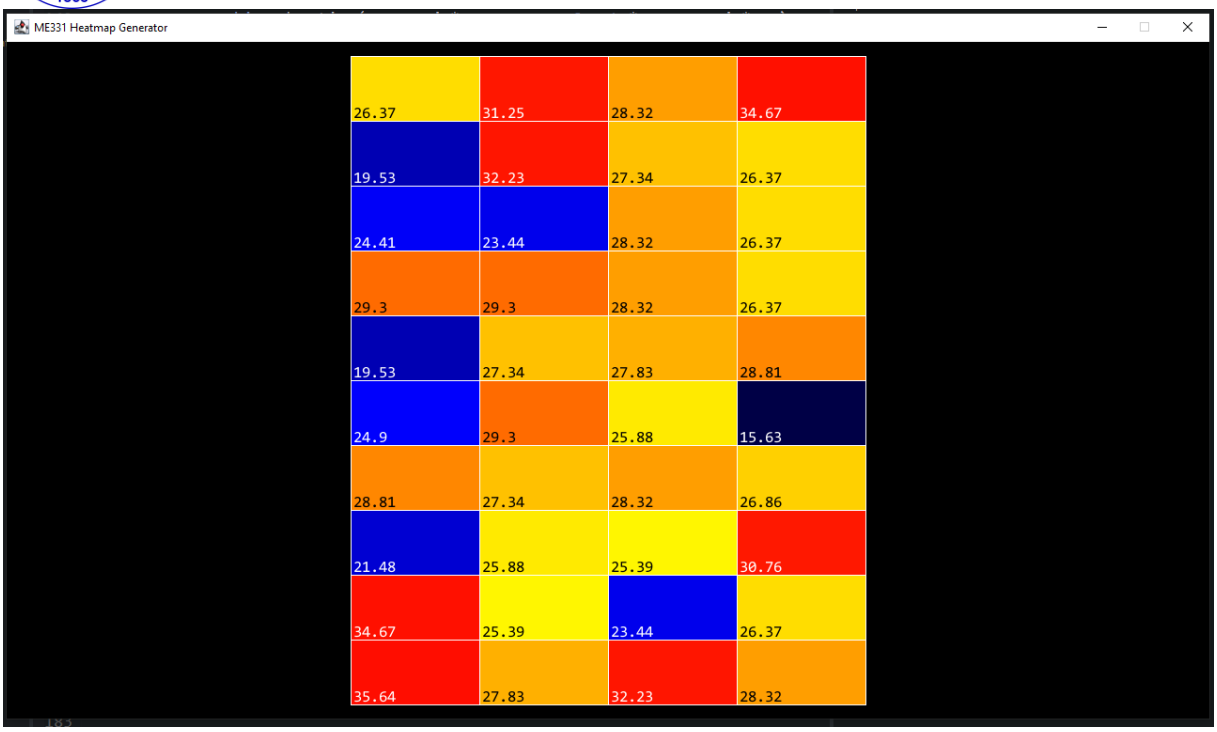
The Algorithm

The embedded software logs the temperatures to a text file with two decimal places. So, the text file is parsed, and the temperatures are recovered.

After that, all the temperatures are divided into ranges of 15-25, 25-30, and 30-40 degrees Celsius. The color for each temperature is interpolated between its range's maximum and minimum colors. This interpolation is done on the linear RGB color space of Java to get better results.

In the end, a JFrame is created with a Canvas inside that can be used for drawing inside. A rectangle for each measurement is drawn with the calculated color and the temperature is written on the bottom left of the rectangle in white or black depending on the brightness of the color.

Also, in each update, the mouse wheel input is tested, and the map is scaled based on that. After scaling the map is centered to the window.



The Development Process

The development of this algorithm started a lot later than the embedded software. There were two main problems encountered along the way.

The first one was reading the temperature data. Initially, it was planned to read and write the input and output data as bytes. But later the embedded software had issues writing and reading bytes.

Debugging that problem took a lot of time, and communication over the internet only slowed this process. So, reading and writing the input and output to and from the Arduino was based on a text format, where the Arduino would parse the input text and print it to the output file. Considering this, the Java program was changed and lots of code was removed.

The second main issue was calculating the colors for temperatures. In the first version, three different colors were interpolated. They represented the minimum, maximum, and average temperatures.

These colors mixed and created unwanted hues. So, they were changed to shades of a single color in the next iteration. This way no weird colors appeared, but it tired the eye of the lookers.

Then different libraries were used to make the interpolation of color better, but they did not make much difference. To fix this issue the temperatures were divided into ranges that will have similar colors. So, the interpolation will be done in a way that would not generate unexpected hues and colors.

8. Embedded Control System Development

by: Cem GEÇGEL

The Algorithm

The robot has 5 states: VERTICAL, HORIZONTAL, ANGULAR, ERROR, and DONE. It works by shifting between them.

Setup

At the set up the robot gets ready to do its task.

1. Set the modes for pins of the drivers and the LEDs.
2. Initialize the MPU library, for the gyro. If it fails, go to ERROR state.
3. Calibrate the gyro and set its threshold.
4. Initialize the SD library, for the SD card. If it fails, go to ERROR state.
5. Read the variables given from the SD card. If it fails, go to ERROR state.
6. Write the variables to the output file. If it fails, go to ERROR state.

Update

The robot is updated in a variable time step loop.

1. Save the current time.
2. Calculate the current yaw using the gyro.
3. Set the angle as the difference between the aimed yaw and the current yaw.
4. Calculate the current position by estimating the speed.
5. Update the state or blink the red light if in ERROR state.
6. Calculate the elapsed time using the previous time.

State VERTICAL

The robot moves through the waterway, and it writes the temperature occasionally.

1. Set the driver pins so that the robot goes forward.
2. If the position is bigger than the length of a row, set the aimed yaw according to the turn direction. Set the position and the speed to zero. switch to the ANGULAR state with the aimed state as the HORIZONTAL state.
3. Else if the distance after the next measurement position is positive write the current temperature to the SD card.

State HORIZONTAL

The robot shifts from a waterway to the next one.

1. Set the driver pins so that the robot goes forward.
2. If the position is bigger than the width of a row, set the aimed yaw according to the turn direction. Set the position and the speed to zero. switch to the ANGULAR state with the aimed state as the VERTICAL state.

State ANGULAR

The robot turns a right angle around its center.

1. Set the driver pins so that the robot turns around its center in the counterclockwise direction if the angle is positive or vice versa.
2. If the absolute value of the angle is smaller than the threshold change the state to the aimed state.

State ERROR

The robot encountered an internal problem, and it waits while blinking the red light.

State DONE

The robot completed going around the whole farm. Now it waits.

The Development Diary

After the algorithm was developed the next step was to implement this in code. This process required a lot of research and trials.

Development started with a PIC in mind, but later it created too many problems than it solved. So, it was replaced by an Arduino.

For this project two drivers, an SD card reader, and a temperature sensor were required. Most of the research was around using them with a PIC controller. But this became easier after switching to an Arduino.

After the algorithm, the controller, and the components to use were generally decided, the development turned to a mostly iterative process. The number of different versions of the code got out of hand so much that a source control tool became necessary.

The biggest change in this time was the addition of the gyro. Open-loop control of both the position and the angle of the robot proved to be an issue. Because of this, it was decided to close-loop control the angle. This not only improved the turning precision but also helped to correct the forward direction of the robot.

The below table shows this process in detail. It is color-coded. Blue changes are solely for debugging, while the green ones mean a fix of an issue. The orange changes are the ones that broke the code and most of the time generated an issue.

Date	Version	Changes	Problems
28.01.2021	0.1	Implemented the previously created logic in the C code for PIC.	Could not implement the drivers, the temperature sensor, or the SD card in PIC.

	1.0	Rewrote the code for Arduino.	Still, drivers, the sensor, and the SD card were not implemented.
3.02.2021	1.1	Implemented the driver to control the wheels.	The sensor and the SD card were not implemented.
6.02.2021	1.2	Implemented reading to and writing from the SD card.	The temperature sensor still was not implemented.
		Reads and writes floats as 4 bytes.	
		Changed the step size from a constant to a variable given by the user.	
	1.3	Changed the row width to a float from an unsigned char.	
9.02.2021	1.4	Implemented the temperature sensor.	The code was not tested. There were not any known problems at this point.
	2.0	Commented out the SD card and the temperature sensor implementations to only test the driver implementation.	The wheels were tested alone without the robot. They would rotate to make the robot go forward as expected. But they would not make the robot turn right angles at the corners and just stop at them.
	2.1	Changed the driver pins.	

2.2	Made the left set of wheels rotate in reverse, thinking they were incorrect.	
2.2	Changed the signal strength to full, thinking that was the problem that stopped the wheels from rotating.	
2.3	Fixed the left wheels back to how they were. They were not the problem.	The wheels did not rotate at all.
	Changed the forward signal to zero. So going forward and turning corners were isolated from each other.	
2.4	Changed the test row length to zero. So the robot would not try to go forward at all and only turn the corners.	
2.5	Changed the turn method parameter to an unsigned char so it matches with the turn signal. Fixed the turning issue.	In the state where the robot should turn a right angle around its axis, the wheels rotated in unexpected directions.
2.6	The forward signal and the row length were changed back to their original values as the problem was fixed.	
2.7	Adjusted the expected angular speed constant by trial.	
	Added a red and a green LED to debug the state of the robot.	

2.8	The LEDs were set to show if the robot is trying to go forward or turn. Also, they change with the turning direction.	
2.9	Changed the turn signal to an int from an unsigned char. As negative signals were needed for the clockwise turn.	The robot changed its turning direction at the wrong times. Its initial turn was wrong and it would stop later than it should. Also, the robot would not stop after completing its job and would turn in its end spot forever.
2.10	Changed the check for the end of the row and the turn direction change to the correct state.	
2.11	Adjusted the expected angular speed constant by trial.	The robot changed its turning direction at the wrong times. Its initial turn was wrong and it would stop later than it should.
	Carried the check for the end of the row to the incorrect state.	
	Added signaling the wheels to stop at the end of the program. Fixed the robot turning at its end spot forever.	
2.12	Changed when the check for the end of the row and the turn direction change was done. Thinking that was the issue.	
2.13	Swapped both checks, fixing the turning direction and breaking the end of the row check.	
2.14	Fixed the direction of the initial turn. It is reversed at the end of the first row. So, it should be the reverse of the input.	The robot changed its turning direction at the wrong times. It would stop later than it should.

2.15	Adjusted the expected angular speed constant by trial.	
	Carried both checks to the end of the row. Fixing the issue with the turning direction completely.	No glaring issue was found other than the angle the robot would turn to be too variant.
1.15	Changed the driver pins and the signal strengths so they match what we used in the tests.	All of the bugs fixed in the driver tests were not changed in this version. All of the problems are still here.
3.0	Took the main version 1.15, but commented out the user input parts to only test writing to the SD card.	The computer would not recognize the SD card. So, there are no tests here.
3.1	Added reading and writing ints as 4 bytes too.	
	Implemented the fix from version 2.15.	
3.2	Changed how an int is written as 4 bytes.	
3.4	Implemented the fix from version 2.5 and 2.9.	
3.5	Added a red and a green LED to debug the state of the robot.	

10.02.2021		Implemented the fix from version 2.11.	The SD library would not initialize.
	3.6	Implemented so that the output file is opened and closed after each time it is written to.	
		Fixed the debug lights so they close after the program is done.	
	3.7	Started to use GitHub for version control. Amount of different versions surpassed 30 at this point.	
		Set the LEDs so they show the error in the SD card implementation.	
	3.8	Tried setting the chip select pin of the SD card explicitly to 10.	
	3.9	Omitted the not operator while checking if the SD library initializes, thinking that causes the issue.	
		Removed all of the unused parts of the SD implementation for clarity.	
	3.10	Changed the chip select pin to 4.	

3.11	Changed the chip select pin back to default (10).	
	Changed the shield voltage to 5V from 3.3V. Since it already has a voltage regulator on it.	
3.12	Tried setting the chip select pin explicitly and writing digital HIGH to it manually.	
3.13	Tried the same thing in version 3.12 with pin 4 instead of 10.	
3.14	Removed everything else completely other than the SD library initialization.	
3.15	Removed all the checks related to the initialization of the SD library and opening of the file.	
3.16	Added a LED back to see if the code still worked.	
3.17	Added an output that writes to the SD card at each iteration of the loop.	No apparent problem with the SD card implementation at this point.
	Formatted the SD card. Fixed the issue where the library would not initialize.	

	1.17	Implemented the fix from version 2.15, 2.5, and 2.9. Added the int reading and writing from version 3.2.	Some fixes were not implemented yet.
	1.18	All of the fixes other than the 2.5 from the test versions are added.	Forgot to fix the bug that was fixed in the test version 2.5.
11.02.2021	2.16	Took the main version 1.18, but removed the SD card parts to only test the drivers.	The wheels would not rotate at all.
	2.17	Adjusted the expected angular speed constant by trial.	
	2.18	Removed everything else completely other than a constant counterclockwise turn. Connected the STBY (standby) pin to 5V. Fixed the no wheel rotation issue.	The robot inconsistently turns around its axis. The dead reckoning for the current yaw is not working well enough.
	2.19	Changed the code so that the robot turns for a second in both directions.	
	2.20	Increased the time to 5 seconds with 3 seconds stops in between them.	

2.21	Changed the code so that the robot turns for 36 seconds clockwise.	
2.22	Tried decreasing the signal so the robot turns slower.	
	Accidentally made the robot go forward after the turn.	
2.23	Fixed the unwanted forward movement.	
2.24	Added a ten-second-long delay after the setup so there is time to put the robot down after loading code to it.	
2.25	Increased the signal strength to full again. Lower speed made the variation in the angular speed worse.	
2.26	Decreased the delays in the test so they can be done faster.	
2.27	Added the code from version 2.16 back, but with a different expected angular speed constant after the tests.	The robot stopped turning.
	Lowered the significant figures of the analog to celsius constant to 6. So, it is in the range of a float variable.	

	Reintroduced the bug that was fixed at version 2.5 while bringing the code from version 2.16.	
2.28	Fixed the expected angular speed constant. It was calculated wrongly.	
2.29	Implemented the fix from version 2.5.	The robot inconsistently turns around its axis. The dead reckoning for the current yaw is not working well enough.
2.30	Adjusted the expected linear speed constant by trial.	
1.9	Same as version 2.29 the fix from version 2.5 was implemented again.	No apparent problem at this point.
1.10	The change in version 2.24 was added to the main branch because it is very useful.	
2.31	Added a full stop before and after turns to decrease the variation on the actual angular speed of the robot.	The robot continued to go forward after the stops and did not turn at all.
2.32	Changed the pin parameters of the driver method to ints from chars.	The robot inconsistently turns around its axis. The dead reckoning for the current yaw is not working well enough.
	Fixed the full stop.	

2.33	Removed the full stop as it did not do any change.	
2.34	The full stop was added again.	
	Adjusted the expected angular speed constant by trial.	
2.35	Different expected angular speed constants are created for both the clockwise and opposite directions.	
2.36	Adjusted the expected angular speed constant by trial.	
2.37	Adjusted the expected angular speed constant by trial.	
2.38	Made the robot just turn and skip everything else. So, the turning is on the lens.	
2.39	Implemented a gyro.	No glaring issue was found other than how the robot's yaw changes more than 1 degree while moving forward.
	Turned the main loop to a fixed time step loop.	

1.11	Merged all of the test versions to the main one.	
	Added define guards for every different feature so they can be turned on and off at will.	
	Commented off the define guard for logging. So driving part is isolated.	
1.12	Created another define guard for reading input from a file. Then commented it off.	The robot stops after logging the first temperature.
1.13	Added serial logs to understand what is happening.	
1.14	Added even more serial logs.	
1.15	Added more serial logs and removed unnecessary parts to clean up the code.	
1.16	Stopped writing the temperatures as bytes. Now the output file is in text format.	
1.17	Added a stop before reading temperature.	

	1.18	Added serial logs around the temperature logging to understand what is happening.	
	1.19	Added serial logs around the delay that makes the loop fixed time. It is suspected to be the cause of the issue.	
	1.20	Turned the main loop to a variable time step loop. Fixed the stopping after the first temperature reading issue.	No glaring issue was found other than how the robot's yaw changes more than 1 degree while moving forward.
12.02.2021	1.21	Created another define guard for writing to the serial output.	The robot would not move forward between corners. It just turns again. It would not change the turning direction. Also, it would not take any temperature measurements.
		Made the turning signal power linearly change with the distance from the desired angle.	
		Stopped calculating the position twice for measurement. And made the speed a variable. So, it can be adjusted.	
	1.22	Added serial logs for the angle and the given turn signal thinking it was the problem and the robot turns 180 degrees.	
	1.23	Added serial logs for the raw yaw calculated using gyro's output.	
	1.25	Added a lower bound on the turn signal calculation. Thinking it is the problem.	The robot would not stop to turn and wobble around 90 degrees. It would not take any temperature measurements.

1.26	The robot would turn until the signal reaches 0. The lower bound stopped this so turns never got concluded. Fixed.	The robot would not move forward between corners. It just turns again. It would not change the turning direction. Also, it would not take any temperature measurements.
	Lowered the lower bound to 100 from 150.	
1.27	Changed the relationship between the angular difference and the turn signal to linear between angles square.	
	Increased the lower bound to 125.	
1.28	Changed the relationship between the angular difference and the turn signal to linear between angles square root.	
	Increased the lower bound to 155.	
1.29	Added more serial logs to the end of the turns.	
	Increased the lower bound to 200.	
1.30	Fixed the turn direction not changing.	The robot would not move forward between corners. It just turns again. Also, it would not take any temperature measurements.

	Changed some of the serial logs to make them easier to read.	
1.31	Changed the logs on the movement to see the actual position than the position after the turn was initiated.	
1.32	The expected speed was non-zero while the robot was turning. This caused it to not move forward. Fixed by assigning 0.	The robot would stop turning before it reaches a right angle. Also, it would not take any temperature measurements.
	Lowered the lower bound to 125 from 200. After realizing that was not the cause of the issue.	
1.33	Increased the lower bound back to 200.	
	Changed the relationship between the angular difference and the turn signal back to linear.	
1.34	Increased the lower bound to 255. Like before, the robot is turning at a fixed speed. This fixed the incomplete turning.	The robot would not move the same distance when it goes and comes back. This shifts it around 26% of the distance each time.
	Now the angle is checked against a threshold to conclude the turn not the signal against the lower bound.	
	The logging was done when the distance after the point was negative. Now it is done if it is positive. Fixed the issue.	

	Changed so the forward signal would go down on one side to keep the robot on track. This fixed the robot yaw changing.	
1.35	Lowered the tolerance values down to 0 for both turning and going forward.	The robot would not stop to turn and wobble around 90 degrees.
1.36	Increased the tolerance to conclude turning to 0.01 degrees. Fixed the wobbling.	The robot would not move the same distance when it goes and comes back. And shifts around 26%.
1.37	Multiplied the expected linear speed by 0.26 when it comes back.	
1.38	Changed the multiplier to 0.80. Expected speeds matched with real speeds. Fixed the dead reckoning for linear movement.	No glaring issues were found at this point.
1.39	Turned reading back on using the define guard.	The input bytes could not be converted to floats correctly.
1.40	Applied a mask before shifting bytes when reading them.	
1.41	Used unions to read bytes.	The input bytes could not be converted to ints or floats correctly.
	Added a red and green LED which indicates error or data logging.	

	1.42	Because reading failed, writing tried to see if that worked.	The output bytes are not sensible to read ints or floats from.
	1.43	Changed the byte variable in the unions to unsigned chars from uint8_t's.	
	1.44	Read the lines as strings and convert those to floats or ints, and print results. Fixed the handling of input and output.	No glaring issues were found at this point.

9. Electronic Systems

by: M. Çağatay Sipahioğlu

9.1 Electronics Preface

After agreeing upon the scope of the project, we need to equip a mechanical skeleton with a suitable brain and nervous system. In this context we need to choose a suitable Microcontroller, (MCU from now on) the most common and easy microcontroller that is used is the Arduino MCU (rather the AVR series ATMEL MCU inside it). Arduino as a board has some very nice features on board like a voltage regulator that eases use. But in my opinion the most important part of the Arduino environment is the software and libraries which makes it significantly easy to code. Thanks to these libraries users don't have to work with Special Function Registers, Configuration Bits etc. Although this eases prototyping endeavours significantly, it also abstracts the users from the inner working principles of the MCU development. Since we wanted this project to be as much of an educational experience as well as a practical one, we will be using a PIC MCU for the development of this project as long as feasible. Lastly it must be noted that the PIC development environment is not as supported as the Arduino environment in Turkey (Although more supported than other MCU lines like ARM) and this will impose some limits on how this project can be developed in this relatively limited environment, and these limits will be discussed as encountered with solution suggestions.

I have tried to give as many sources I could as to wherever I found the information I have written but I also must give some background. I have been around robots a lot in high school thanks to our school club, mainly mini-sumo and line-follower robots. I had a general experience in the working principles of the robots and I had production skills such as soldering and PCB manufacturing but I wasn’t involved in the electronic and software part of the development process, there was always someone who did that for our club. I for example knew that I should use PROTEUS if I wanted to simulate circuits but didn’t know how to use it coming to this project. I knew that I should be looking at the “datasheets” to build the circuits but I hadn’t even opened a datasheet coming to this project. And finally I hadn’t touched any PIC coding or programming coming to this project.

9.2 PIC Selection

In order to select a PIC we should be eliminating unsuitable PICs from the producer's catalogue and we should be left with a number of options with slight variances to choose from. It should be noted that PIC selection will be made with considerations regarding the other modules that will be used. Which modules are chosen to be used are explained in detail in the "Part Selection" section but here is a brief list of limitations imposed by the peripheral modules in order to help us eliminate unsuitable candidates:

- Temperature & Gyro Sensors: PIC should have at least 3 ADC modules.
- SD Card Module: PIC should support SPI communication. (used for data transfer to SD Card) PIC should also have at least 2kB of RAM, SPI is rather RAM consuming on its own and the end result will probably use most of these 2kB, therefore ideally RAM should be more than 2kB.
- Motors & Actuators: PIC should support at least 3, at most 6 pwm outputs. (Considering differential drive for each of the 4 motors and 2 reserved for the use of actuators)

When we look at the catalogue there are a lot of features/peripherals for each PIC, in order to choose a suitable PIC from amongst hundreds we first need to understand what each feature is needed for. Below I have researched and explained nearly all of these features in order to choose the correct PIC. The modern PIC catalogue and the more inclusive MAPS (Microchip Advanced Part Selector) can also be accessed from the links below:

Modern Catalogue: https://www.microchip.com/ParamChartSearch/Chart.aspx?branchID=1005
MAPS: https://www.microchip.com/maps/microcontroller.aspx

PIC Selection Criterias

- **Family:** Family mainly separates MCU products from the microprocessors. (MCP from now on) We will be looking for a microcontroller therefore we will use Family=MCU. There are also 8,16, 32 and 64 bit MCUs. These bit numbers correspond to the data, address and register bus widths, which simply adds up to how much data you can manipulate at once. Higher bit numbers result in more data manipulation and more instructions. For comparison an Arduino uses an 8 bit MCU, PC RAMs are usually based on 32 bit MCPs and 64 bit MCPs are used in computers and supercomputers. 8 and 16 bit MCUs are more suitable choices for practical applications like ours where not that much data has to be manipulated, they also are more cost effective as expected. In the end if the amount of data that will be processed is very high 16 and 32 bit MCUs can be considered. Between 8 and 16 bit MCUs 8 bit will be more beginner friendly and also will be the financially responsible choice. It should also be noted that from a practical standpoint for many small to medium applications like ours, 8 bit MCUs are plenty enough and the usual choice.
  Verdict: Use an 8 bit MCU.
- **CPU Type:** This parameter separates PIC and AVR MCUs. AVR series for example are used in Arduinos (ATMEL Subseries) and along with PICs they are widely used in applications like robotics. Apperentaly AVR offered lower power consumption but it seems that PICs now also do. Both types should able us to do many things we want to do in this project but we should opt for the use of PICs simply because more models are available in Turkey. Other than that we always have Arduinos if AVR ever needs to be used.
  Verdict: Use a PIC series MCU.

- **Max CPU Speed:** CPU speed is as it says speed. Our application isn't that time sensitive but it would always be better to not be slow at the cost of price of course. A medium value like 20-40 MHz can be chosen considering both speed and price.
  Verdict: 20-40 MHz is a good choice interval.
- **FPU:** FPU stands for floating point unit, this unit is useful for float operations. It is not offered with any models after applying the previous limitations. It is a rather specialized unit that is not critical for our application.
- **Operation Voltage Min & Max:** An operation voltage of 1.8V-5.5V is standard for most PICs it is a reasonable interval for powering with 3 AAA batteries or much better a 1S Li-Po battery or 2-4S Li-Po battery with a voltage regulator.
  Verdict: 1.8-5.5 V
- **Temperature Range Max:** PICs usually offer 85-125 degrees celsius of maximum value. Since our application is planned to be used in at most half of those values these values are not a dealbreaker.
  Verdict: Not an eliminating factor.
- **Pin Count:** Usually around 7 pins are already occupied with power (4-VDD,VSS,AVDD,AVSS), crystal (2 pins for OSC) and MCLR (1 pin) pins. On top of that for our application we will need at least 8 at most 14 pins for driving motor control. (8 if the two right motors will be controlled from the same signal, 14 if each motor will be controlled independently.). Probably at least 3 more for actuator control. (PWM+2 Directional Pins). 1 pin for temperature sensor, 2 for gyroscope, 4 for SD Card. And an extra 7 pins for a debugging LCD and a debugging switch would be nice to have. In total there should preferably be over 38 pins. Around 38 PICs usually have either 40 or 48 pins both would suffice also considering future upgrades.
  Verdict: 40 or 48 pins.
- **Program Memory Size:** For a medium length program like ours around 20kBs of program memory should be needed. In order to be on the safe side 24+kB of memory should suffice. (20,24,28,32 kB etc. are offered in the catalogue)
  Verdict: 24+kB.
- **SRAM:** Static RAM or SRAM is where the variables are stored. The SD Card module in our application will hog a significant amount of RAM since it works with a significant data buffer and variables. +512 bytes are recommended just for SD Card communication. We may estimate in total 1 kB of RAM will be used for the rest of the program. We should also leave around maybe %25 of the memory empty in order to not run into any complications 2+ kB of program memory will be ideal.
  Verdict: 2+ kB SRAM.
- **EEPROM:** EEPROM (Electronically Erasable Programmable Read-Only Memory) is where we would store data permanently. In theory the EEPROM module can do the job of the SD Card for data storage in our application. It should be noted that even the bigger EEPROMs usually offer around 1kB of storage. It would be a limiting factor if we had decided to store all data in the EEPROM since the amount of data can vary greatly with the amount of sensors and the size of the fields that are being traversed. So we will be using the SD Card in order to not be limited in case of scaling. Therefore EEPROM will not be used and is not needed.
  Verdict: No need.

- **Low Power:** This property apperentaly used to be offered mostly for the AVR series but right now each PIC in the new catalogue offers this. This property is especially useful for applications with a battery, like ours and will be good to have.
  Verdict: Yes.
- **ADC (Analog to Digital Converter) Input:**Many of our sensors will be producing analog data. We need 2 for gyroscope, 1 for the temperature sensor. At least 2-3 more are needed for pH and water sensors. (future improvement) All in all 6+ ADC channels should suffice considering even further future additions.
  Verdict: 6+ ADC Input.
- **DAC (Digital to Analog Converter) Output:** It is not usually offered, and offered in low quantities. As far as I can say DAC is used to produce specific signal profiles for precise electronics applications. Since PWM modules are discussed elsewhere I don't think this makes a difference for a practical mechanical application like ours.
  Verdict: No comment.
- **UART, SPI, I2C Modules:** These are all communication modules, usually used for Master-Slave applications, in order to communicate between 2 PICs or with sensors or actuators using the same communication protocol. For example SD Cards use the SPI protocol and at least 1 SPI module is needed for this reason. Gyroscope sensor needs an UART or I2C module. And the LCD communicates on I2C protocol. Therefore at least 1 of each is needed, more would be appreciated in case of future improvements.
  Verdict: 1+ UART (or USART), 1+ SPI, 1+ I2C.
- **Max PWM Outputs:** Considering 4 motors and at least 1 pwm for actuator (Assuming the use of a servo motor for moving the probe into the earth) 5+ PWM outputs will be appreciated. (3+ can be used if needed by parallel connecting the 2 right motor inputs.)
  Verdict: 5+ (3+ in worst case scenario)
- **Number of PWM Time Bases:** We don't need different time bases since we will use PWM's duty cycle simply for powering and 1 time base . More time bases would be needed for precise electronic wave generation.
  Verdict: Unimportant.
- **Packages:** Package type changes how the PIC will be mounted on to the circuit and in some cases the strength of the package material. The main difference is between DIP (Dual In-line Pin) and SMD (Surface Mount Devices) packages, actual package codes may differ. DIP packages have mechanical legs that let them to be mounted on breadboards, PIC programmers or sockets. SMD packages must be soldered on to the PCB in order to be used and comes in much smaller sizes, SMD packages utilizes ICSP (In circuit serial programming) for initial programming. SMD packages also can have more pins due to no mechanical restrictions. (It would be a nightmare to try to insert a 100 legged DIP package) Some SMD packages like the BGA for example have small connection points below the PIC which are soldered by smearing the pads with solder paste and heating the PIC with hot air after placing it, melting the solder below and connecting the PIC to the circuit. This package is used in smart phones for its extremely compact characteristic. For our application we will be using a DIP package since it can be prototyped on breadboards and programmed directly. SMD packages are also not compatible with our project at the moment since we won't be producing a PCB.
  Verdict: DIP Package.

Final Selection (PIC18F47Q43)

- After applying the above restraints, four very suitable choices remain: PIC18F46Q43, PIC18F47Q43, PIC18F56Q43, PIC18F57Q43. Among these the latter two have 48 pins which may be a lot more than we want to use although still suitable. Nevertheless opt for the former 2 40 pin PICs. Among these two PIC18F47Q43 has bigger program memory and SRAM therefore is the better choice (Although the former will also suffice.) As a result for the case that this project is made with a PIC a PIC18F47Q43 should be used. The elimination process can be seen in attachment: PIC Catalogue.xlsx
- PIC18F46Q43 Characteristics: 64 MHz Max CPU Speed, 1.8-5.5 V operation range, 40 pins, 128 kB program memory, 8kB SRAM, 35 ADC Input, 5 UART, 2 SPI, 1 I2C, 6 PWM.
- **Problem:** None of these PICs can be found in Turkey. Although PIC18F47Q43 should be used for the main application we are not able to work on that PIC at this time. Nevertheless we should be able to experiment with PICs at this time, therefore a second catalogue elimination was conducted this time for the PICs available in Turkey. PIC18F4431 was chosen and purchased in order to start development. This PIC has 6 PWM channels and offers SPI for the SD Card module.
- **Important Notice for Development:** Although we have found a PIC that we can start working on it will not be enough to handle the whole project on its own. (Due to RAM limitations with the SD Card module) The ideal choice of: PIC18F47Q43 would be enough. But this is no reason to stop learning how to use a PIC. We should be learning as much as we can to make this project into a reality with a PIC but when it comes to demonstrate the algorithmic code we have written we should use an Arduino since it will have enough resources. From now on developments on the PIC are aimed strictly towards the ideal form of this project and for education purposes. And Arduino should be utilized to demonstrate that the algorithmic code can and is written which will let us control the robot as we want to. And this way we will be able to show that if the PIC18F47Q43 could have been provided we would also be able to transfer the same algorithms and be able to use it even with a PIC.

9.3 Module Selection

This part explains the thought process behind choosing which modules to use and which modules not to use. And discusses types of these modules that can be found in the market.

- **Motors:** In order to move the robot we need tires or tracks attached to tires. We are aiming to build a robot that is not too big so it can drive in the waterways of the field, then we also won't need motors that are too big but we will need motors with high torque since we will need to drive in rough terrain. Considering that the robot probably won't be too big or too heavy but still might need torque when necessary, motors used in the mini-sumo field should be providing enough torque. 12-16mm electric DC motors will probably be sufficient. (Exact values should be derived from calculations) These motors have around 1A maximum current draw. And since torque is our main concern over speed the motor should have a reductor. For these small motors lower rpm values usually correspond to higher torque values therefore we should look for a somewhat low rpm motor.
  Verdict: 12-16 mm with reductor.
  Final Choice: 6V, 180 rpm, 12 mm motor with reductor: https://www.robotus.net/high-torque-6v-180rpm-mikro-reduktorlu-dc-motor-mini-sumo-robot-motoru
- **Encoder:** Encoders are attached to motors and are able to determine how much the motor is turning. Encoders can be used to calculate the displacement of the robot through angle calculations of the tire and construct a closed loop algorithm. However in our working environment, which is dirt, the tires might turn without moving the robot in patches of loose dirt. Therefore encoders won't be a reliable denough way to establish closed loop control.
  Verdict: Not reliable enough.
- **Motor Driver:** There are a few motor drivers that are used a lot like the L298N and L293D. But from these two, L298N is awfully inefficient (It is better as a radiator then a motor driver) and L293D can't output more than 650mA which is below our needed current value. L293 is a good contender with 1A max current output when needed but upon further research there is an even better option which is the TB6612FNG motor driver. TB6612FNG offers an additional standby pin for motor control, making the robot stand at the ready for movement, which is a plus over the L293. It also offers a 1A steady output rather than a peak output over the L293 which is much better and safer. TB6612FNG is a very good and sufficient motor driver for our application. All the mentioned drivers can drive 2 motors each.
  Verdict: TB6612FNG x2
- **Battery:** The main user of the battery power will be the motors PIC or the Arduino will reduce the voltage before usage. Although the motors are listed as 6V, in their description it is stated that they can be used with up to 12V and with voltages lower than 6V as well. Because the torque values of the motors are so high we will get away with using not the total 12 Volts. Also not pushing the motors to their max will be healthier. Considering all these, a battery with 6V+ of voltage will be good. TB6612FNG also has a working range of 4.5-13.5 Volts therefore our battery should be 6-13.5 Volts.
  For robotics applications Lithium-Polymer batteries (Li-Po) are widely used for their sufficient power capability, small size and efficiency. (Li-Po are more efficient than the Li-Ion batteries we see everywhere for example) Li-Po batteries start from 3.7 volts which is 1 cell or 1S and go up by multiples of 3.7, so they go up in voltage as 7.4 (2S), 11.1(3S) etc. For our application

we should opt for a 7.4 volt Li-Po since we won't be using the whole 11.1 Volts if we had used a 3S.

The mAh value of the battery will be determining the maximum operating time of the robot. We should first consider our power consumption. Our main consumer will be the motors since MCU uses much less power compared to the motors and therefore can be neglected in calculations. Assume that we are using %75 of the motor power on average. Which is 0.75 A/motor. Then we would be drawing 3A at 5 Volts. (We actually tested these motors on the robot in the debugging week and the motors can even be driven and carry the robot with a maximum of 5 Volts (PIC or the Arduino's operating voltage)) Therefore our power is 15 Watts. A battery has its capacity in Watt hours, a 2S 1350mAh battery has the Watt.h value of nearly 10 W.h (7.4 * 1.350=9.9 Wh) which means that this battery can supply 10 Watts for 1 hour or 1 Watt for 10 hours etc. For our application this battery would result in a 10/15=40 minutes of operating time. For an aim operating time of 1 hour we need to increase the battery's mAh value by %50 and by this calculation we will need a 2S 2025 mAh Li-Po battery with the assumption that motors are working at %75 power in average. The closest Li-Po being produced to our aim has mAh value of 2200 mAh which will give us 1 hour of operating time, hopefully cancelling out the inefficiencies with the extra mAh value on top of 2025.

$$Operating\ Time = \frac{Battery\ Power.h}{Consumption\ Power.h}$$

$$= \frac{Battery\ Voltage\ *\ Battery\ Ah\ Value}{Operating\ Voltage\ *\ Average\ Operating\ Current}$$

$$1hour = \frac{7.4\,Volts * Battery\ Ah\ Value}{5\,V * 3\,A} \qquad Battery\ Ah\ = 2Ah = 2000mAh$$

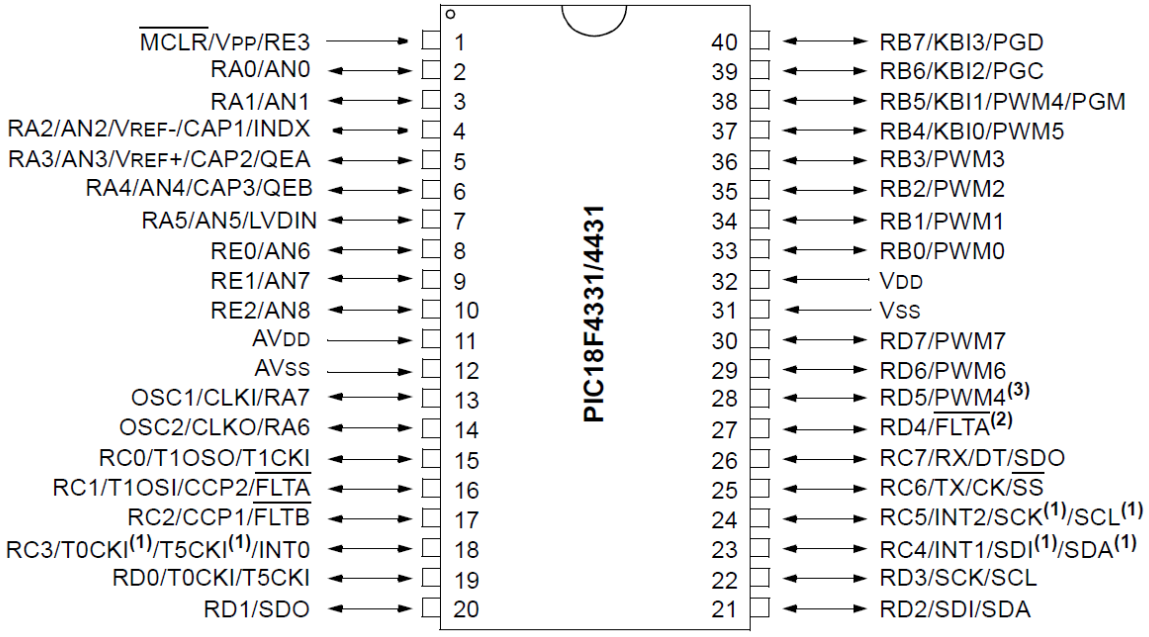
Verdict: Li-Po, 2S (7.4V), 2200 mAh.

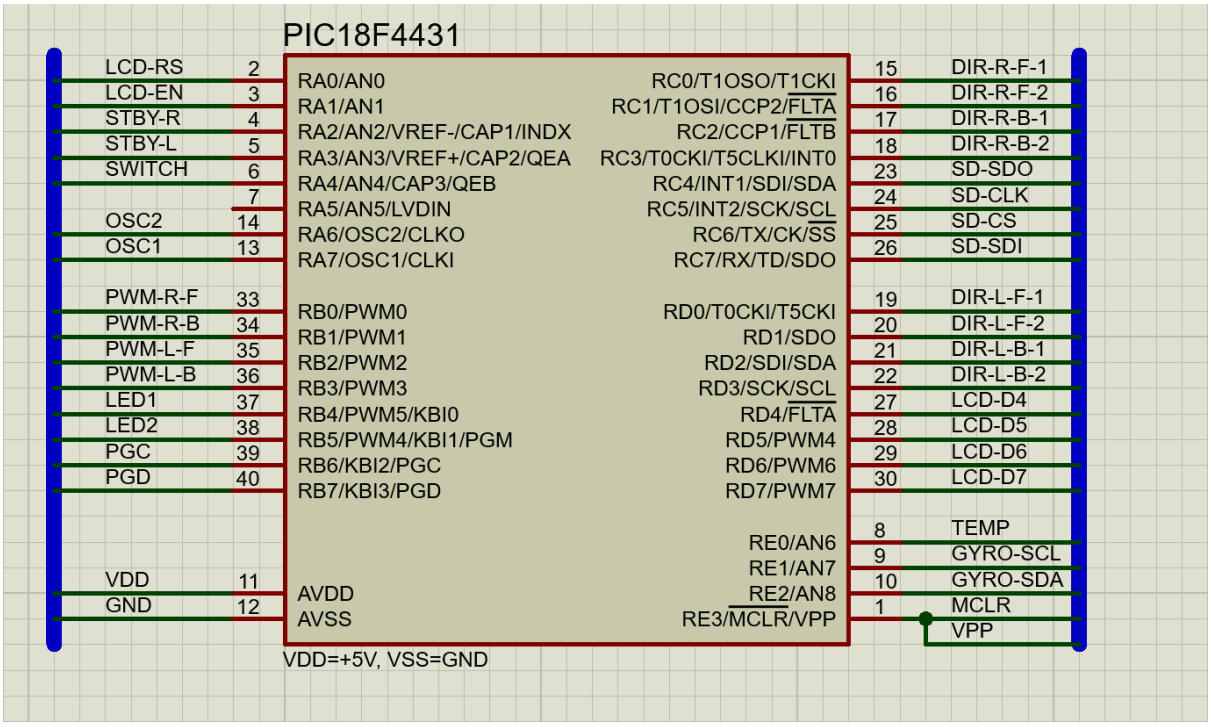
- **Voltage Regulator:** In order to supply the PIC or the Arduino with 5 Volts we need a voltage regulator that will reduce 7.4 Volts to 5 Volts. We can use an L7805CV which can take a voltage value from between 7-35V and reduce it to 5Volts.
  Verdict: L7805CV
- **Actuator:** The actuator should have a part that can probe the earth and dissolve the dirt in water in order to measure its pH with the builtin sensor. And also a specialized part that can measure the dampness of the soil. Designing such a part could be a project topic on its own, therefore we will just be simulating the use of the actuator via a LED.
- **SD Card Module:** We need to use an SD Card module in order to write and read on and from the SD Card. Although we could directly connect to the pins on the card itself to achieve this, it would be nice to have a dedicated slot that we can plug the card in. There are SD Card modules sold that accomplishes this, we should also use one of these. If you look at the module you can see some circuitry on it, the circuitry is for an on-board voltage regulator and an IC that reduces the data pins to 4. That regulator is there because SD Card operations are conducted at 3.3 Volts (not our operating voltage of 5V) and thanks to the regulator we can power the module with 5Volts like the rest of our circuit and not worry about frying the card.
  Verdict: Use SD-Card Module: https://www.robotistan.com/sd-kart-modulu-sd-card-module

- **External Oscillator:** An external oscillator can be used with a PIC to determine the instruction cycle frequency. For any PIC Instruction Cycle Frequency= Oscillator Frequency/4. For this reason you might see 4 MHz crystals (oscillators) used quite frequently since that means that the instruction cycle frequency is 1MHz which corresponds to 1 instruction per $1\mu s$. 1 Instruction cycle corresponds to a single line of Assembly Code being conducted. Therefore for a 4 Mhz crystal a line of assembly is executed every microsecond. PIC18F4431 for example has a maximum CPU speed of 40Mhz but also supports lower speeds when used with the appropriate crystal. (See circuitry for connections) In order to use this PIC at 20Mhz we need a 20Mhz crystal oscillator connected properly to the PIC and this way can use the same PIC but much faster. (I believe the default internal oscillator of this PIC is 4Mhz. Note: Not all PICs have internal oscillators, these PICs will have to be connected to an external oscillator to work all together) The reason we are not using the PIC at 40 Mhz is because it is not achieved directly through a 40MHz crystal, rather it is achieved through using a 10MHz crystal with PLL (Phase Locked Loop) enabled. And since we wouldn't be able to easily debug if something went wrong with PLL we will use a 20 MHz crystal, still plenty fast for our application.
  Verdict: Use a 20MHz crystal oscillator.
- **GPS Module:** Another contender for us to achieve closed loop control. However we need precision nearly down to a centimeter and the best any commercially available GPS can offer is around a meter, therefore not quite good enough for our purposes.
  Verdict: Not precise enough.
- **Analog IR Sensor:** In another attempt to achieve closed loop control the robot may be outfitted with IR distance sensors on its sides and could try to center itself with respect to the plants on its right and its left in an attempt to stay true to the waterways. Although in theory it sounds good, in reality more than likely the amount of gaps between the leaves or the bodies of the saplings may prove too large or inconsistent for the robot to use them as reference.
  Verdict: Doubtful, may be used if other methods fail to keep the orientation true.
- **Gyroscope:** A gyroscope module will be able to give us the orientation data of the robot along its axis and this will allow us to correct our course since we know the orientation of the paths within the field. This module is a small but very effective solution to our orientation problems, letting us achieve closed loop control for the orientation of the robot in the field. (It will calibrate itself to 0,0,0 rotation at its start position.) We can use the MPU6050 3 axis accelerometer and axis gyroscope. It is cheap yet highly effective for our purposes. Also the orientation of the robot on the X and Y axises can also be used to map the elevation differences and uneven parts in the waterway, which can then be used to correct water flow in the field.
  Verdict: Definitely should be used, Use MPU6050.
- **Temperature Sensor:** Although MPU6050 has a crude temperature sensor inside we will use a seperate precise temperature sensor in order to collect temperature data around the field.
  Verdict: Use LM35DZ Temperature Sensor
- **ICSP:** Stands for In Circuit Serial Programming. It is basically 2 pins reserved for ICSP that can be used to program the PIC without removing the PIC from the circuit. It is more of a reservation than a module. We should reserve and connect a 5 pin male header to our circuit in order to be able to use ICSP when needed. (See circuitry) (5 pins consist of 2 reserved data and clock pins and 3 power pins)

9.4 Circuitry (PROTEUS)

I will be using the circuitry and simulation program PROTEUS in order to draw the circuits and simulate whenever I can. I will be building the circuit around PIC18F4431 since PIC18f47Q43 isn't supported since it is a somewhat newer model. But the building and circuitry principals will stay the same therefore the knowledge can be applied along. First of all before starting to draw the circuit we should get to know our PIC a little better. Below is given the PIC and its pins and this will be our reference throughout the circuitry process. I will usually refer to VDD/Operating Voltage as +5V but the PIC would work in its specified voltage range. Operating voltage is connected to pin 31,32.

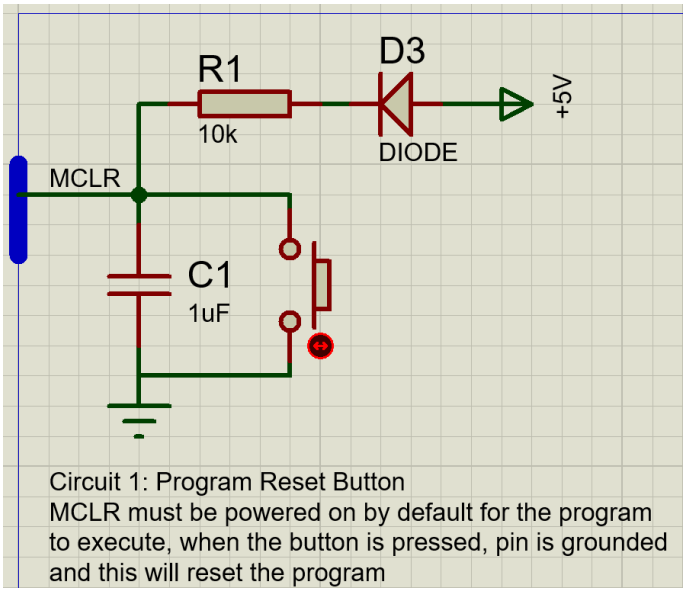




Blue lines here are actually the bus functionality of the PROTEUS program but in my use they can be thought of as portals, since it lets us connect any 2 wires with the same name through the blue buses.

9.4.1 MCLR Circuit

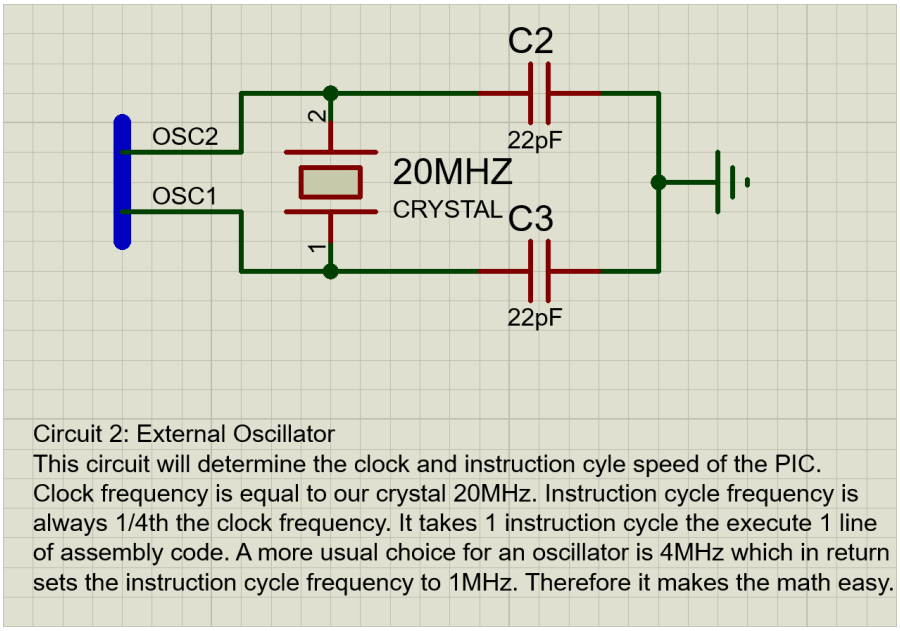
In order for a PIC to work the MCLR pin (Pin 1) must be connected to the +5Volts. When this pin is grounded the program resets. We can put a button between this pin and ground in order to have a "reset button" since when we press it MCLR pin will be grounded and the program will reset. This is the same working principle as the reset button in the corners of Arduinos. I have also put a diode between the +5V incase of faulty powering. The resistor and the capacitor protect this pin against the voltage spikes or noise that may be coming from the power source. And I will also put a capacitor parallel to the button so the voltage change isn't too sudden when resetting and so this won't damage the PIC.



Circuit 1: Program Reset Button
MCLR must be powered on by default for the program to execute, when the button is pressed, pin is grounded and this will reset the program

9.4.2 External Oscillator Circuit (20 MHz)

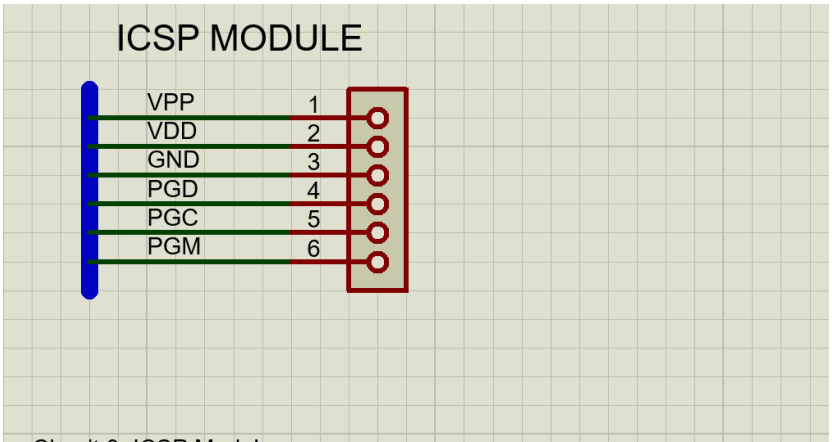
In order to run our PIC at 20MHz we must connect an oscillator with 2 22pF capacitors in the configuration below. How to connect this circuit is shown in the datasheet of the PIC. (Page 29) This is also how an external oscillator is connected to any PIC. External Oscillator must be connected to the OSC1 and OSC2 pins of the PIC. (13,14 for this PIC)



Circuit 2: External Oscillator
This circuit will determine the clock and instruction cyle speed of the PIC. Clock frequency is equal to our crystal 20MHz. Instruction cycle frequency is always 1/4th the clock frequency. It takes 1 instruction cycle the execute 1 line of assembly code. A more usual choice for an oscillator is 4MHz which in return sets the instruction cycle frequency to 1MHz. Therefore it makes the math easy.

9.4.3 ICSP Connections

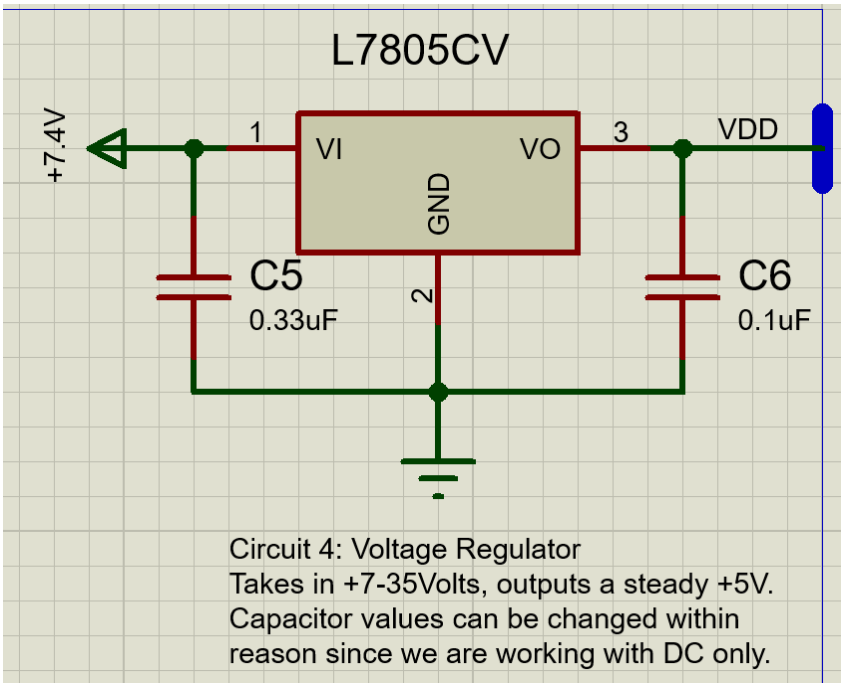
When using ICSP very high voltages are used to program the PIC, these voltages act on VPP which is also the MCLR pin which is Pin1. VDD & GND are the +5V connections of the PIC. PGD (Pin 40) is for Data transfer, PGC (Pin 39) is for the clock signal. PGM is used for low voltage programming and not used by us. PGC and PGD pins are critical and should be reserved since otherwise there may occur programming errors.



Circuit 3: ICSP Module
ICSP=In Circuit Serial Programming. ICSP lets us to program the PIC without removing it from the board. It is beneficial to add an ICSP connection point to our circuits for this reason unless we need all the pins on the PIC for some other reason. The actual programming in the ICSP is conducted on the PIC programmer. This part in our circuit is basically a port to connect a cable between the programmer & the PIC. Note that PGM (AUX) is not used.

9.4.4 Voltage Regulator (L7805CV)

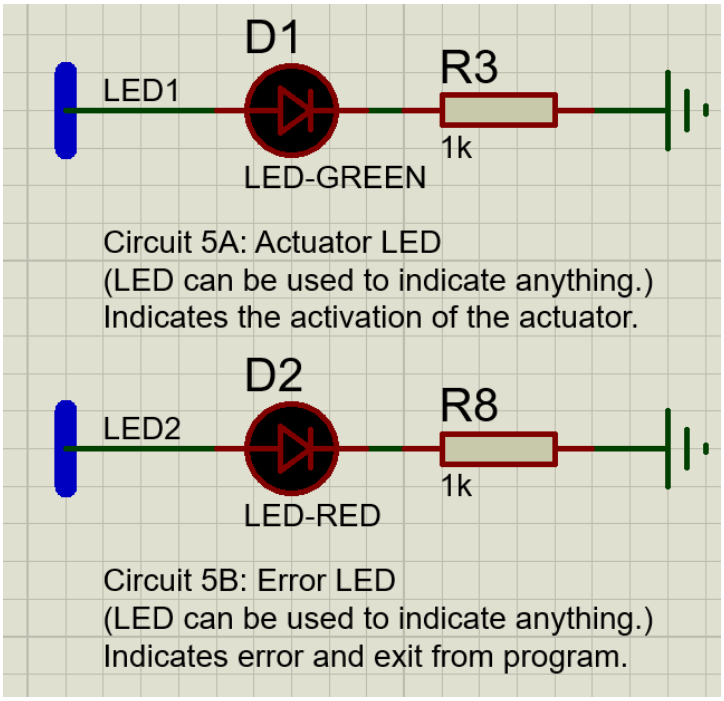
We need to regulate the Li-Po voltage down to +5V for the use by PIC. L7805CV Integrated Circuit (IC from now on) does precisely this job. In the lower configuration it outputs a steady +5V from an input of +7-35 Volts. The circuitry can be looked up from the datasheet of L7805CV. (Page 22) This IC outputs +5V in this configuration denoted with the 05 in its product code.



Circuit 4: Voltage Regulator
Takes in +7-35Volts, outputs a steady +5V. Capacitor values can be changed within reason since we are working with DC only.

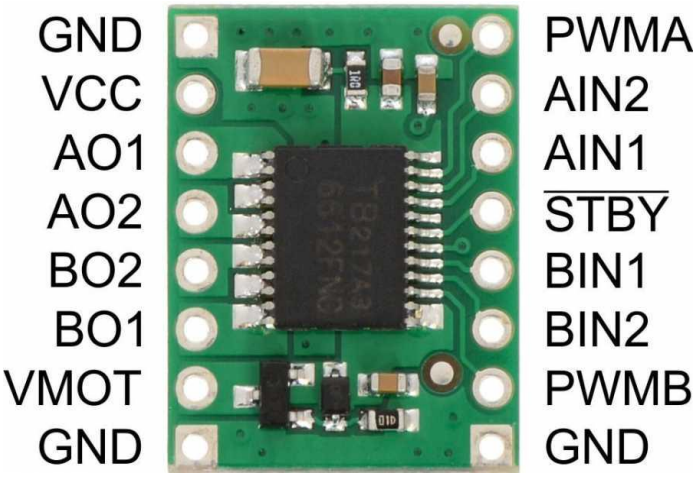
9.4.5 LED Circuits

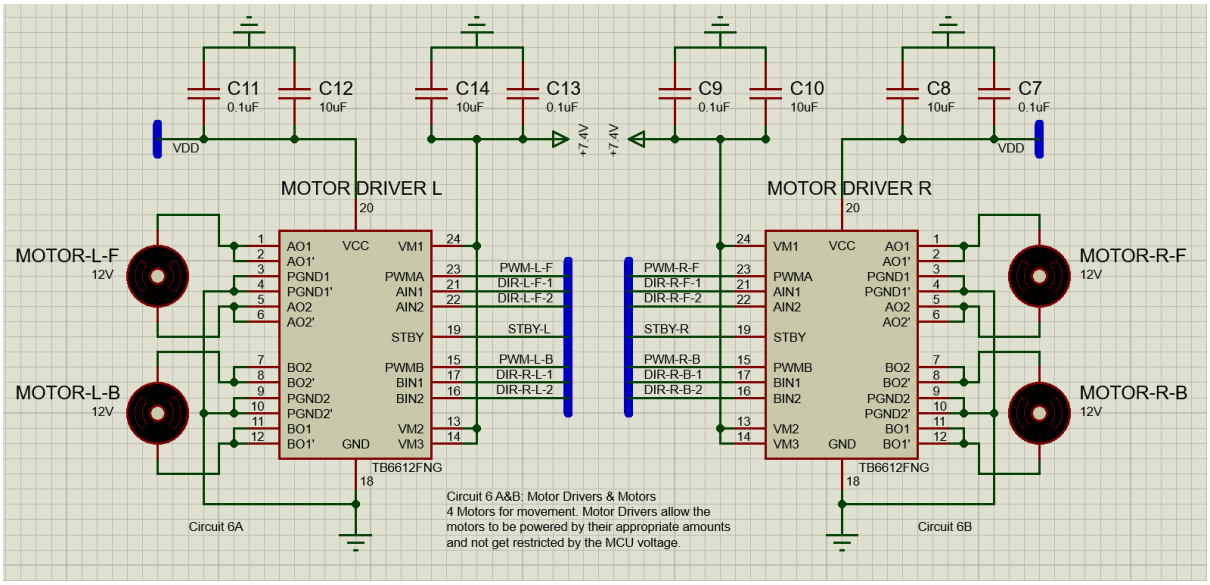
Every circuit needs LEDs for something. The circuitry is very straightforward, just an LED with a resistor, getting power from pins 37 or 38. In our case red LED will be used to indicate error and the green LED will be used to show that the “future actuator” is working.



9.4.6 Motor Driver Circuits (TB6612FNG)

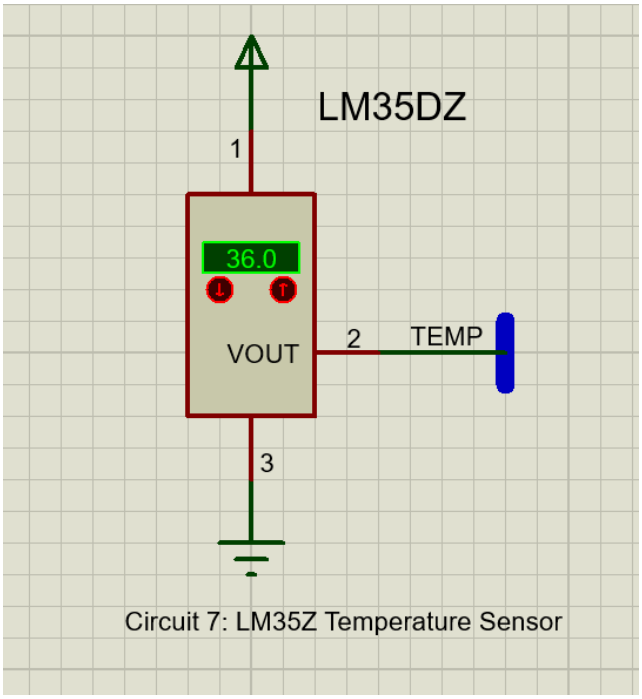
These circuits have been built to be thorough with respect to the specifications in the TB6612FNG datasheet Page 7. If the TB6612FNG is bought as a separate IC these circuits must be implemented in order for the drivers to be functional. But usually these circuits are sold built as a module and used by just giving in inputs and taking out the outputs. 7 inputs (2 pwm, 4directions, 1 standby) and 4 outputs (2 for each motor) + 4 power lines (2 for circuit power, 2 for motor power (Vmot, connected directly to the battery)) Motor control occupies most of the pins on the PIC from 4-5, 15-18, 19-22, 33-36 (PWM)





9.4.7 Temperature Sensor (LM35DZ)

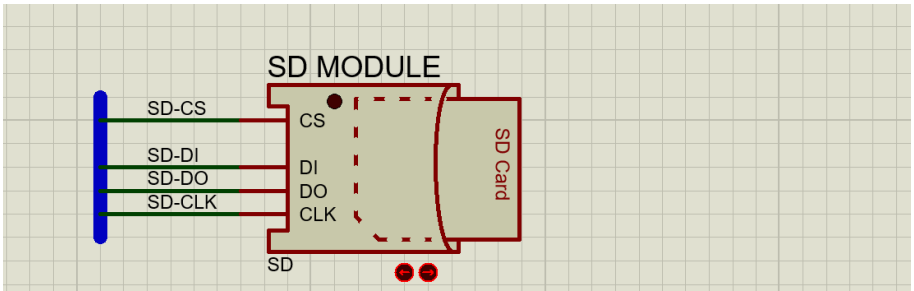
LM35DZ outputs the temperature data such that 10mV of output corresponds to 1 degree celsius. With an operation voltage of +5V we can map the input value by multiplying the input value (0-1023) with $\frac{500}{1023}$ which will become 0-500 in order to measure a range of 500 celsius. (LM35DZ has upper limit of 110 C°) For example 36 degrees celsius will result in a 360mV output, which will be implemented as (0.36/5)*1024=73.728 out of 1023 as an analog input. $\frac{73.728*500}{1023} = 36\ C°$ This analog data is sent to Pin8. Notice that Pin8 has AN functionality for Analog to Digital Conversion. Also in order to do ADC on the PIC we connect AVDD and AVSS to the VDD and VSS lines. AVDD and AVSS will be taken as reference voltages when declaring appropriate registers in the programming part.



Circuit 7: LM35Z Temperature Sensor

9.4.8 SD Card Module

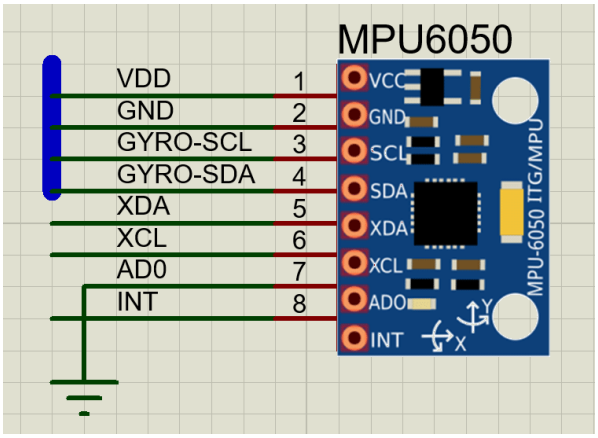
SD Card Module has 4 pins for SPI data transfer. These pins must be connected to their SPI counterparts on the PIC. Firstly the CS (Chip Select) pin of the SD Card must be connected to the SS (Slave Select) pin of the PIC. (Pin 25) The CLK (Clock) pin of the SD must be connected to de SCK (Serial Clock) pin of the PIC. (Pin 24) The DI (Data In) pin of the SD must be connected to the SDO (Serial Data Out) pin of the PIC. (Pin 26) The DO (Data Out) pin of the SD must be connected to the SDI (Serial Data In) pin of the PIC. (Pin 23) For other master-slave applications MOSI (Master Out Slave In) and MISO (Master In Slave Out) abbreviations might be used to denote the complimentary naming of Pins 26 and 23 respectively.



Circuit 8: SD Card Module.
Collected temperature, pH and water data will be stored in the SD-CARD since the on-board EEPROM of the PIC is not nearly big enough. SD CARD modules works on 4 pins with SPI communication protocol. PIC18F4431 has 1 SPI module which will be used for this purpose. SD-CS pin denotes the chip select pin, which is connected to the SS (Slave Select) pin of the PIC. SD-DI (Data In) and SD-DO(Data Out) pins are connected to the SDO (Serial Data Out) and SDI (Serial Data In) pins of the PIC respectively. For this reason, sometimes MISO (Master in Slave Out) and MOSI (Master Out Slave In) abbreviations are used in order to denote SD-DI and SD-DO lines in the circuit.

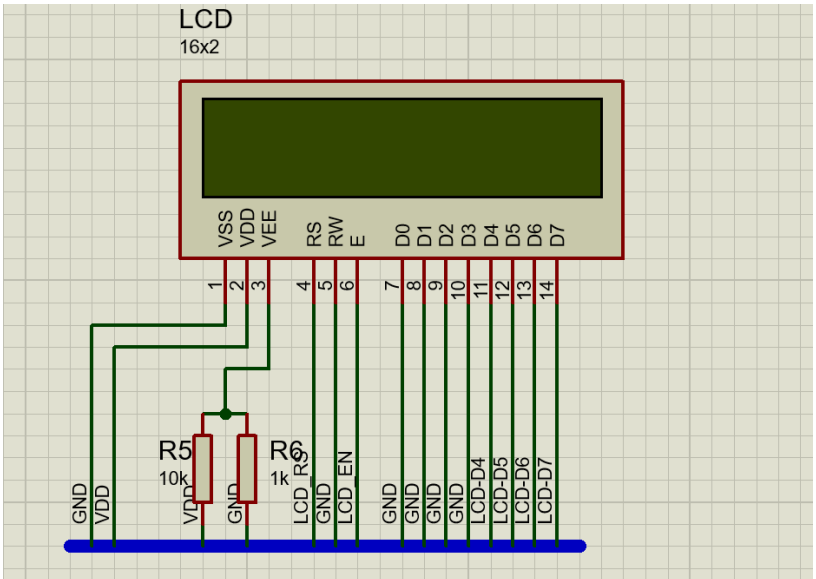
9.4.9 Gyroscope Module

Sorrily there is no library for MPU6050 in PROTEUS therefore I will draw the connections here but we won't be able to simulate it in PROTEUS. This module gives the orientation of the robot in three dimensional space with respect to initial calibration. This module can and will be used for the measurement of the rotation of the robot along its center axis. This module will be useful to go straight properly and also will be very useful when trying to turn for 90 degrees properly. Since otherwise we would have to use dead reckoning and assign some amount of time where the robot rotates around its axis. But this amount of time and power will nearly definitely result in different angles of rotations each time. Note that the use of SDA (Serial Data) and SCL (Serial Clock) only is sufficient for data transfer; XDA and XCL are for auxiliary connections, INT is the interrupt pin and AD0 is the I2C address select pin.



9.4.10 Debugging LCD 16x2

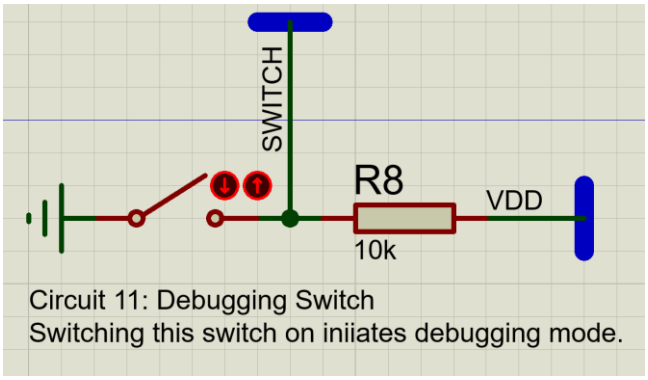
For the debugging mode we should put in a 16x2 LCD, which will give us real time feedback. We will use the LCD in 4bit mode therefore we ground D0 to D3 pins. We also can ground the RW pin to keep the LCD in read mode. (Will be reading data from the PIC) Lastly, at the VEE (Contrast Adjust) pin there usually will be a potentiometer in many example circuits that can be found online but it is a waste to use a potentiometer for this application since we won't be changing the contrast after finding a value that is suitable. For this we can build a makeshift potentiometer with 2 resistors connected to power and ground and taking the constant contrast signal from in between them. In order to find a value we like we can use a potentiometer for one time only and find a contrast value we like, we can then measure the resistance at that instant and replicate the same value with resistors. The resistor values below should work just fine too, if a calibration wants to be skipped.



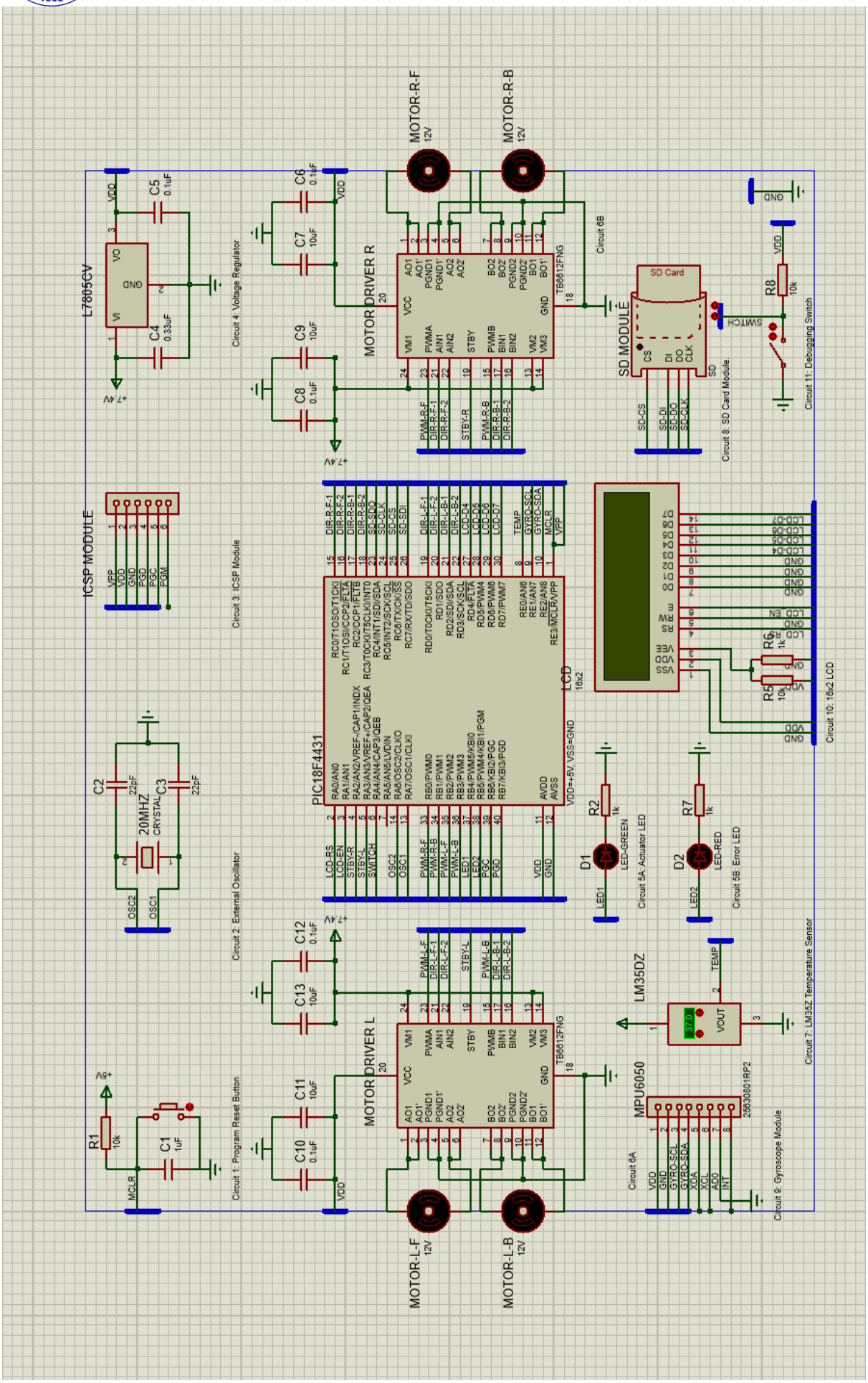
Circuit 10: 16x2 LCD
Useful for Debugging. Can be used to see many aspects of the code and pinpoint errors.
For VEE contrast setting, usually a potentiometer is used, however two
resistors can be made into a makeshift potentiometer with the configuration
above, after calibrating the contrast to our liking with appropriate
resistance values it is much better to use this configuration so we won't
need a potentiometer every time we want to use a LCD.
RW pin is grounded which fixes it to read mode. (Reads from the MCU).
D0-D3 are grounded since we will be using the LCD in 4 bit mode.

9.4.11 Debugging Switch

In order to enter debugging mode we should put in a switch. When the switch is switched on we enter debugging mode, when it is off we will run the normal program. This way for example we can turn off the debugging LCD when we aren't using it.



Circuit 11: Debugging Switch
Switching this switch on iniiates debugging mode.


PIC18F4431
LCD
16x2
MOTOR DRIVER R
MOTOR DRIVER L
TB6612FNG
MOTOR-R-F
MOTOR-R-B
MOTOR-L-F
MOTOR-L-B
SD MODULE
SD Card
ICSP MODULE
L7805CV
LM35DZ
MPU6050
20MHZ
CRYSTAL
LED-GREEN
LED-RED
SWITCH
Circuit 1: Program Reset Button
Circuit 2: External Oscillator
Circuit 3: ICSP Module
Circuit 4: Voltage Regulator
Circuit 5A: Actuator LED
Circuit 5B: Error LED
Circuit 6A
Circuit 6B
Circuit 7: LM35Z Temperature Sensor
Circuit 8: SD Card Module
Circuit 9: Gyroscope Module
Circuit 10: 16x2 LCD
Circuit 11: Debugging Switch

9.5 PIC Programming

At the point we started implementing the general C code into the PIC we have realised that there would be some major issues rooting from the use of PIC and MPLAB (IDE we were using at the time) some of which we now solved and some of which we used alternative solutions. The main problem was that the very good PWM PIC we are using won't have enough RAM left enough to conduct SD Card communication or there was absolutely no easy way for us to write a working SD code since MPLAB was not supporting SD Card communications with any library for the PIC we are using whatsoever. We would have to spend every second of free time we get in order to develop a full duplex communication code with SPI for the SD Card and even then we would still have to magically create the original PIC we have chosen in the beginning which will have enough RAM to both run a self correcting driver code, an ADC code and a SD Code. Therefore we did the only logical thing and implemented the code in Arduino. Therefore the embedded code division turned their attention to the Arduino development part and I have continued on with the writing of the fundamental PIC programs. After all the choice of using a PIC was never about being able to finish the project solely on PIC, it was about getting down to the roots of circuit and code development for microcontrollers and therefore PIC should continue on serving this very purpose. All in all we have pivoted from PICs because we couldn't get the newer models (For a reasonable price or time for that matter) that would allow us to conduct our code without restrictions and I have continued on to write or learn the fundamental codes so that we would be ready for the ideal self of this project if we have the resources. (Main difference being the use of Special Function Registers in coding) You will realise that whilst developing this project we have noted and commented on many points which can be improved given different resources and time. Just because this project has a window of 2 months doesn't mean that we must treat it as whatever we can put together in two months.

9.5.1. PIC Programmer Selection (K150 PIC Programmer)

There are a few programmer options in the market developed by some companies. (Usually companies that produce IDEs for PIC also produce the programmers so that the user can be supported by a uniform environment.) When trying to buy a programmer in Turkey you will more than likely be met with 3 options. The first is “MikroProg” programmer produced by Mikroelektronika (Developer of MikroC IDE) that is rather rare and priced unreasonably. Second is the “PicKit” programmers produced by Microchip of which newer versions are pricier and PicKit3 is always out of stock. (Original producer of PICs and the developer of MPLAB IDE) And a K150 programmer that wasn’t developed by any big brand and is relatively cheap. All these will accomplish the same job so it made sense to buy the K150 which is about 1/10 to 1/20th the price of its alternatives.

K150 comes equipped with a ZIF socket (You physically insert the PIC into this socket and connection is made with the programmer circuit) or an ICSP connection in order to program the PIC. Many sellers also provide a link or a zip file including the small programming software. This software just detects if the programmer is on, then you can upload the hex file of the code into the software and the software will transfer the program to the PIC through the programmer. However when you plug in the programmer Windows straightaway gives the error: “PL2303HXA PHASED OUT SINCE 2012. PLEASE CONTACT YOU SUPPLIER” As you might expect this is rather problematic we are trying to use a hardware with drivers outdated 8 years and a software which was probably updated last for Vista. There are a few updates that have been rolled out since 2012 for the driver but none of them makes the programmer software able to communicate with the programmer. After hours of trying to fix it I have found the solution in the 2 video links below. It turns out that the driver which phased out in 2012 works fine with the 2010 version but even then, the software will be shooting out errors although it is managing to program the PIC. As a lesson I do not recommend the K150 programmer to anyone but still here is the solution to make it work:

https://www.youtube.com/watch?v=RDdpByMeOPI&ab_channel=LuisAlbertoBorjaG%C3%B3mez
https://www.youtube.com/watch?v=YwOcBT2aHjo&ab_channel=LuisAlbertoBorjaG%C3%B3mez

9.5.2 Programming Language & IDE Selection (MikroC)

Language-wise all IDEs use a C based language with only slight variations regarding the Configuration Bits or bitwise access to special function registers.

IDE-wise there are 3 popular options: MPLAB X (Microchip), MikroC (MikroElektronika) and CCS C (CCS) All have many resources and functionalities that eases the use of PICs when needed. We had started off with the MPLAB X IDE since it was developed by Microchip which also develops the PICs. I have successfully written a code for a simple digital output and I have managed to use the SFRs of the PIC18F4431 to use its power PWM mode to give the desired output signal. (Pretty proud of that one) So we started trying to find a library for the SD Card communication. MPLAB X IDE actually offers a pretty nice solution to this through its Code Configurator. This code configurator or MCC is introduced as working with 18F PICs but apparently not all of them including the PIC we are trying to use the library with. Apparently MCC works with somewhat newer models. After that I have looked at how people used to conduct SD communication before MCC was introduced since MCC is somewhat of a new feature. I found that there are PIC18F peripheral libraries which included the SD Card communication functionality but the new problem was these libraries were now legacy and didn't work with the current compiler. They somewhy update their compilers and these new compilers are not back compatible. Therefore we are not able to use the libraries that are meant for our PIC because they deemed them legacy and made them practically unusable. All in all I felt like Microchip was updating their environments and making them not back compatible in purpose to incentivise people to buy the newer model PICs and therefore decided to change the IDE at this point. It should be noted that with the ideal PIC we chose this problem wouldn't happen but still, I believe any good software should be back compatible. My next choice was to use the MikroC environment since I saw that they have got an official Multimedia Card Library which can be used in our case. And although CCS C had a library for SD Card communications it felt like MikroC was cleaner therefore I chose MikroC. The following codes were written in MikroC and many libraries make it easier to develop code with PIC. I think it can be said the MikroC is for PIC what Arduino is for AVR.

9.5.3 Special Function Registers (SFRs)

A SFR can be thought of as a readily programmable variable. The name and how they will be assigned values will be provided by the supplier of the microchip in the datasheet. Each SFR is 1 byte long or 8 bits for the 8bit MCU family.

Configuration Bits

This part is for PIC18F4431 only but it is a pretty inclusive PIC therefore it will cover many aspects of PIC programming. Configuration bits were probably the most annoying part before modern IDEs since most of them are defined in a single bit and you must define each bit before even starting to code. In modern IDEs most offer a graphic interface to set up these bits and will offer a somewhat default setting even though you skip the part by just clicking OK. There are 14 configuration bytes in PIC18F4431 ranging from CONFIG1L to CONFIG7H. We will encounter the L, H suffixes more than once. They denote the lower or the higher part of an SFR. Since some SFRs will need more than 8 bits to define and since the remaining bits which will be defined will be a continuation, the SFR name is kept the same but given an extra part. For example a function with a 10 bit resolution will probably be defined by the Lower SFR + the lowest 2 bits of the higher SFR. Configuration bits are used to turn on or off some PIC maintenance functions like the power up timer, the watchdog timer, read-write protection etc. For what each bit corresponds to for any PIC the datasheet of the respective PIC must be used. In the datasheet there will be a part where the function of each bit is explained. Let us take the below example. This table shows us how to define the CONFIG1H register to our needs. This SFR is mainly about the oscillator for which we are going to use a 20 MHz one. Internal-External Switchover mode allows us to switch between internal and external clocks since we want to only use the external 20MHz clock for our application, we want this property to be off, so bit 7 should be 0 as instructed. Fail-Safe Clock Monitor allows the PIC to automatically switch to an internal oscillator if an external oscillator fails. This is, as it says in the name, a fail-safe and beneficial to have, we would like this property to be on and therefore bit 6 should be written as 1. The datasheet says that bits 5 and 4 are unimplemented and should be read as 0. We can see that in these registers some bits are left blank so that the relevant bits can be defined within the same byte. For the first 4 bits we can see that all four together define what kind of an oscillator we will use. LP oscillators oscillate up to 200kHz, XT oscillators oscillate up to 4 MHz and HS oscillators oscillate up to 20 MHz. Therefore the correct mode should be used for different frequency values. It should also be noted that the previously discussed PLL mode can be activated in this register and by using a 10MHz crystal in PLL mode we can achieve the maximum speed of 40Mhz for PIC18F4431. It can also be seen that in order to use the internal oscillator we should activate it from here and then it can be seen on page 36 that the internal oscillator frequency can be set in the OSCCON register. All in all when it comes to using SFRs the datasheet is our bible.

REGISTER 23-1: CONFIG1H: CONFIGURATION REGISTER 1 HIGH (BYTE ADDRESS 300001h)

R/P-1	R/P-1	U-0	U-0	R/P-1	R/P-1	R/P-1	R/P-1
IESO	FCMEN	—	—	FOSC3	FOSC2	FOSC1	FOSC0
bit 7							bit 0

Legend:		
R = Readable bit	P = Programmable bit	U = Unimplemented bit, read as '0'
-n = Value when device is unprogrammed		U = Unchanged from programmed state

bit 7 **IESO:** Internal External Switchover bit
1 = Internal External Switchover mode enabled
0 = Internal External Switchover mode disabled

bit 6 **FCMEN:** Fail-Safe Clock Monitor Enable bit
1 = Fail-Safe Clock Monitor enabled
0 = Fail-Safe Clock Monitor disabled

bit 5-4 **Unimplemented:** Read as '0'

bit 3-0 **FOSC<3:0>:** Oscillator Selection bits
11xx = External RC oscillator, CLKO function on RA6
1001 = Internal oscillator block, CLKO function on RA6 and port function on RA7 (INTIO1)
1000 = Internal oscillator block, port function on RA6 and port function on RA7 (INTIO2)
0111 = External RC oscillator, port function on RA6
0110 = HS oscillator, PLL enabled (clock frequency = 4 x FOSC1)
0101 = EC oscillator, port function on RA6 (ECIO)
0100 = EC oscillator, CLKO function on RA6 (EC)
0010 = HS oscillator
0001 = XT oscillator
0000 = LP oscillator

Some Other Properties That Can be Set Inside the Configuration Bits

- **Power-up Timer:** When enabled this feature doesn't start the program until the oscillator waves get stable enough. Can be thought of as a capacitor for software.
- **Watchdog Timer:** This timer gets updated along with the user's program, whenever the program fails the watchdog timer resets. It can be used for debugging, for example an endless loop error can be detected with the proper use of the watchog timer.
- **MCLR Pin Function:** When MCLR pin is enabled RE3 pin is disabled. MCLR pin can be disabled to use RE3 pin as normal.
- **Brown Out Reset:** Enabling this would able the PIC to reset if the power gets lower than the Brown Out voltage (Usually around 2V), it should be noted that the program resets regardless if the power supply voltage drops below the lower limit i.e. 1.8 Volts.
- **Low Voltage ICSP:** This functionality allows the PIC to be programmed with lower voltages. Needs the additional PGM pin reserved to do it. (We didn't reserve the PGM pin in circuitry since we won't be enabling Low Voltage Programming.)
- **Code Protection:** This makes the program not readable or changeable after being written. Can be used for commercial products where right infringements may occur.

Some Fundamental SFRs

- **0b, 0x Syntax:** When assigning 1s or 0s to the bits of any SFR we will use the syntax SFR=0b_____ or 0x__ . 0b prefix shows that we will be defining the SFR byte as 8 bits and the 0x prefix shows that we will be defining with the hexadecimal equivalent of the byte. Exactly how they are used can be seen in the individual SFR definitions, but as a general example the TRIS SFR can be looked at below.
- **TRIS:** This SFR is used to set each pin as either an input or an output. Each bit is written as a 1 if we want to use the pin as an input and as 0 if we want to use it as an output. A declaration is made in the following form: "TRISA=0b00100001;" this line for example will be assigning input and output characteristics to the A port. 0b in the beginning of the definition refers to the binary number we wrote after it. It can be thought that each pin of this number is corresponding to a pin on the A port. For example the right most bit, which is the 0th bit will be assigning input output values to the A port Pin0 (RA0). With this logic we can see that the above statement is assigning RA0 and RA5 as inputs and the rest of the port A pins as outputs. An alternative assigning method is with the equivalent hexadecimal number. Equivalent of 0b00100001 is 0x21 in hexadecimal, note that we now use the "0x" prefix to denote definition in the hexadecimal number system. Therefore an equivalent declaration would be TRISA=0x21. People who have worked a lot with assembly language coding are usually fluent in these conversions and they would prefer to declare each of these in hexadecimal since it takes less time to write. Similar declarations can be made for the other ports. We should also note that not all ports must have 8 pins, for example PIC18F4431 has 3 pins on the E port. For this we can look at the data sheet to see how we define TRIS for th E port. (Page 124) It can be seen that the 3 valid E pins are defined as the least significant bits of the TRISE register and bits 7-3 are read as zeros. So TRISE=0b00000010 would define RE1 as an input and RE0 and RE2 as outputs.
- **PORT:** This SFR is used to give digital outputs. 1 writes a pin as HIGH and a 0 writes it as LOW. PORT SFR is used similar to the TRIS SFR such that the letter of the port that is being assigned HIGH or LOW values must be added as a suffix. An example declaration of PORTB=0b00000101; will output a HIGH signal from RB0 and RB2 and low signals from the rest of the B port.
- **ANSEL:** There are usually a number of pins that can be used as analog inputs on the PIC. For PIC18F4431 there are 8 analog pins. We will use some of these pins for example to get an analog input from the temperature sensor. All of these pins are multiplexed with their digital counterparts, meaning that we must declare which of the possible pins we want to use as analog and which to leave as digital. Because there are 9 possible analog pins in PIC18F4431 the ANSEL SFR is split into 2 SFRs, namely ANSEL0 and ANSEL1. ANSEL0 controls AN0-7 and the Least Significant Bit (LSB from now on) of ANSEL1 controls the AN8 channel. We can declare ANSEL0=0b0100000 which will let us use the AN6 channel as analog and the rest as digital.
- **ADCON:** This SFR is used to enable Analog to Digital Conversion (ADC) there are many modes to how to convert an analog signal to digital so there exists 4 ADCON registers named from ADCON0 to ADCON3. (Page 240-243) For example ADCON0 manages the enabling and the channel mode of ADC. ADCON0=0b00000001; will for example turn on ADC module in the single channel mode. For example in ADCON1, bits 7 and 6 together define the reference

voltage for ADC. By assigning these bits 00 we define the positive reference as the AVDD pin on the PIC and the negative reference as the AVSS pin on the PIC. This declaration is why we have connected the AVDD and AVSS pins on the PIC in the circuitry part.

- **OSCCON:** This SFR can be used to define the internal oscillator frequency as discussed in the configuration bits part. We can look at page 36 of the PIC18F4431 datasheet to see exactly how it is defined. We can see for example that bits 6-4 define the oscillator frequency and apparently this PIC supports up to 8 MHz of an internal oscillator. We could use a 4 MHz internal oscillator after enabling the internal oscillator in the configuration bits by defining the OSCCON bits 6-4 as 110. So our definition will look like OSCCON=0b_110____.

9.5.4 Fundamental Codes for PIC

Writing code for PIC is very similar to writing code for an Arduino once one understands how to assign SFRs. For example the power pwm mode of the PIC18F4431 is governed entirely by its SFRs. For more complicated tasks such as communication with a Gyroscope sensor or controlling a LCD can be done by hand once the appropriate timing tables for the communication protocols that are given in the datasheet are understood. (For UART, I2C etc.) However for even more complex applications such as SD Card communications we need to be able to manipulate the file system of the SD Card as well as use the SPI communication fully. (Which is FAT32 by default but can be formatted to FAT16 File system) These complex tasks are as it can be anticipated very very time consuming and not worth the time especially when there are libraries available that can do those. For this reason we should use libraries whenever possible. I will be using the appropriate SFRs as specified in the datasheet for the PWM application and if any other application that can be achieved with SFRs after this application we will be able to understand and implement them. However for applications that require a protocol like SPI or UART it is best to leave the hard work to libraries. (Protocols are predetermined ways of sending and receiving data that is agreed upon between the sender and the receiver.)

When coding the LED and the PWM codes I was still using MPLAB IDE and it wouldn't matter at that point since all those programs are achieved through SFRs which are independent of the IDE. However when it came to the SD Card code there wasn't any available library in MPLAB therefore that is when I switched to MikroC. I can say that MikroC is really helpful with its own application libraries such as the MMC (Multimedia Card) library and eases development including these protocols highly. Although MPLAB has MMC solutions as well they are not as widely compatible as the libraries of MikroC. Therefore the codes after PWM were written in MikroC using MikroC libraries whenever appropriate. Also I will only be implementing the fundamentals of the control codes and the fully algorithmic control codes for the robot can be looked at in the "Embedded Software" section. As I have pointed out before by achieving these fundamental codes we can switch the full embedded control code to PIC18F47Q43 which is the ideal PIC if available. In order to test these codes I have used the PIC18F4431 since it is the available choice in Turkey. It must also be noted that PIC18F4431 offers a power pwm mode which is different from the usual method of pwm in PICs. Most PICs offer some CCP (Capture Compare) modules that can be used for PWM output. The use of this module can be achieved through the PWM libraries available in all IDEs. The Power PWM application I will be demonstrating in PIC18F4431 is therefore actually harder to implement than the usual CCP PWM applications.

For each of these codes only the power up timer is enabled differently than the default settings suggested by the IDE for the configuration bits. (In the default settings most of the fancier functionalities are turned off like the watchdog timer or the brown out reset) The TRIS and PORT values are renamed as below

for easier reading of the code, these declarations are used for every code below. (sfr sbit _ at _ is a mikroC syntax which declares names to pins, but as a data type.)

9.5.4.1 LED Code for PIC in MPLAB (Simulation 1: LED, PWM)

For the proof of concept of LED control I will be blinking a LED connected to the RB4 and RB5 pins of the PIC. In PIC programming the “void setup” equivalent of the Arduino is the initial part of the void main code and the “void loop” equivalent of the Arduino is achieved through an infinite while loop inside the void main. (If wanted, a separate void function can be written to write the code similar to the format of the Arduino) Another thing that must be noted is that many IDEs support some sort of a delay function, in MPLAB X this function is __delay_ms(); for delays in milliseconds. In MikroC this function becomes Delay_ms(); and achieves the same functionality. (See Simulation1-LED&PWM

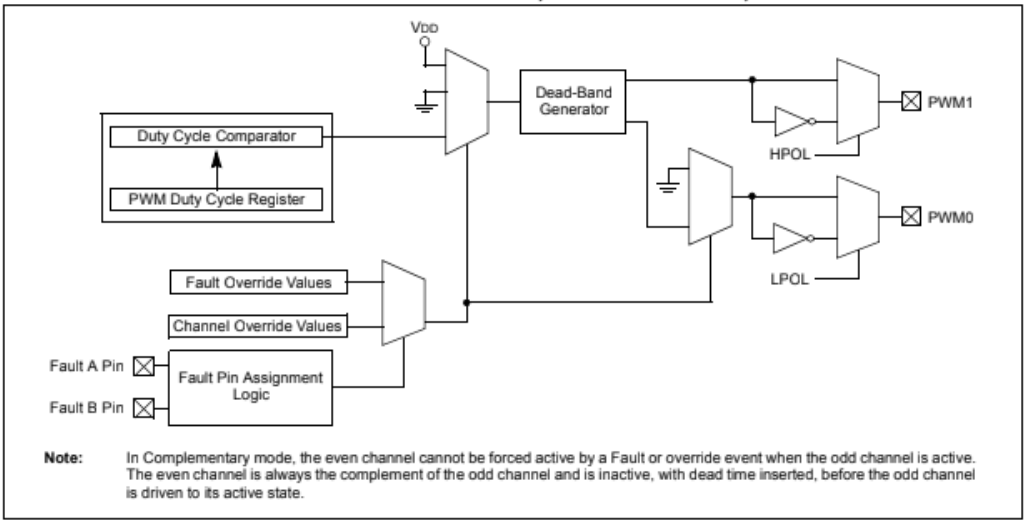
```
void led_blink(){
   LED1=1;       //Proof of concept by blinking the LEDs opposite to each other.
   LED2=0;
   Delay_ms(1000);
   LED1=0;
   LED2=1;
   Delay_ms(1000);
}
```

9.5.4.2 Power PWM Code for PIC in MPLAB (Simulation 1: LED, PWM)

In order to see how we can use the Power PWM mode in PIC18F4431 we can look at the datasheet section 18. (Page 173) The main 4 types of SFRs that are needed to use the Power PWM mode are: PTCON, PTMR, PTPER, PWMCON and PDC registers.

- **PTCON: (PWM Timer Control)** There are 2 PTCON registers named PTCON0 and PTCON1. By reading the datasheet we can see how we can use these registers. PTCON0 register manages the PWM mode, pre and post scalers. The PWM mode that we are used to is called Free Running Mode, the differences with other modes can be seen on pages 181-183. The pre and post scale values will be used to calculate the duty cycle and periods later on. For mathematical ease I will be taking both scales as 1:1. These settings result in the definition PTCON0=0b00000000; for the PTCON0 register. Only the 2 Most Significant bits (MSBs from now on) of the PTCON1 register are used. These two bits define the PWM time base to be enabled and the counting direction. We want to enable the pwm time base therefore bit 7 is a 1 and we want count upwards therefore bit 6 is a 0. (This will make more sense once we understand PTMR) In the end we should define PTCON1=0b000000001;.
- **PWMCON: (PWM Control)** There are 2 PWMCON registers, PWMCON0 manages the enabling and the complementary behaviour of the PWM channels and PWMCON1 manages the special event register. We define PWMCON0=0b00111111. This turns enables the PWM mode for the PWM0-3 channels which are the ones our motors are connected to at the moment. The last 4 ones enable independent mode for each PWM pair starting from PWM0-PWM1. It can be seen on page 175 how the inner circuitry of the PIC allows for independent channels or channel control in pairs (by using the complementary mode, they will be opposite). For PWMCON1 we want to enable PWM override as we change the PWM signal in our program therefore we enable updates and override of the duty cycle in PWMCON1 by PWMCON1=0b00000001. We also put a special event trigger in counting up mode since we are counting up defined in PTCON1.

FIGURE 18-2: PWM MODULE BLOCK DIAGRAM, ONE OUTPUT PAIR, COMPLEMENTARY MODE



Note: In Complementary mode, the even channel cannot be forced active by a Fault or override event when the odd channel is active. The even channel is always the complement of the odd channel and is inactive, with dead time inserted, before the odd channel is driven to its active state.

FIGURE 18-3: PWM MODULE BLOCK DIAGRAM, ONE OUTPUT PAIR, INDEPENDENT MODE

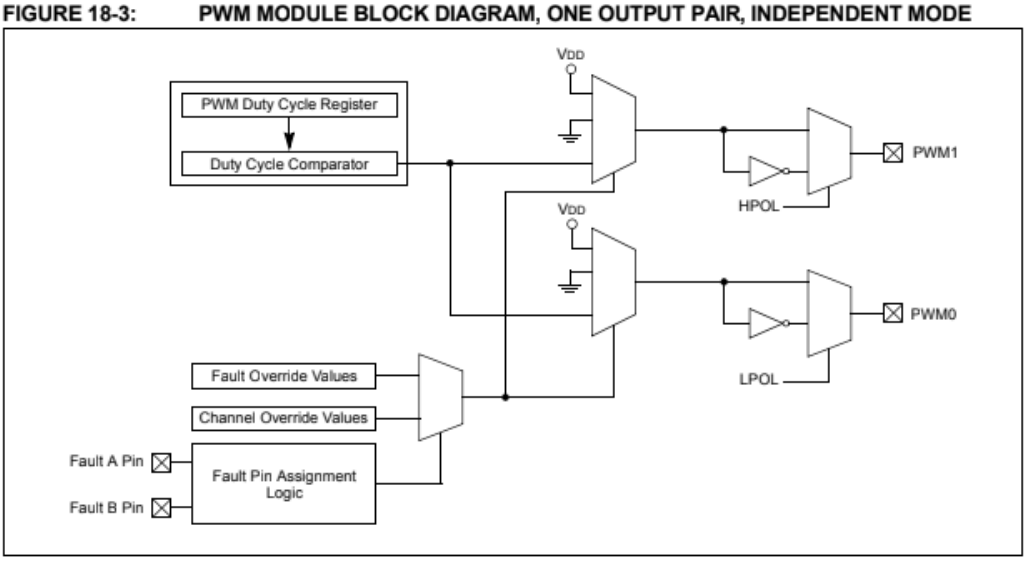
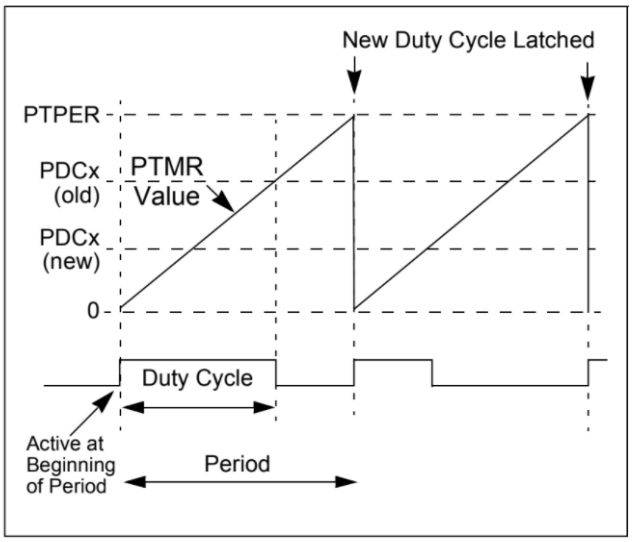


FIGURE 18-12: EDGE-ALIGNED PWM



- **PTMR:** PTMR register keeps track of the current place in the PWM cycle, it resets to 0 once it reaches PTPER. As long as PDC>PTMR duty cycle occurs. Since PTMR is the counter we should initialize it to 0 at the beginning. PTMR uses 11 bits, 8 of them defined in PTMRL and 3 in PTMRH. We initialize all to 0. PTMRL=0x00; PTMRH=0x00;

EQUATION 18-1: PWM PERIOD FOR FREE-RUNNING MODE

$$T_{PWM} = \frac{(PTPER + 1) \times PTMRPS}{F_{OSC}/4}$$

EQUATION 18-2: PWM PERIOD FOR UP/DOWN COUNT MODE

$$T_{PWM} = \frac{(2 \times PTPER) \times PTMRPS}{\frac{F_{OSC}}{4}}$$

The PWM frequency is the inverse of period; or:

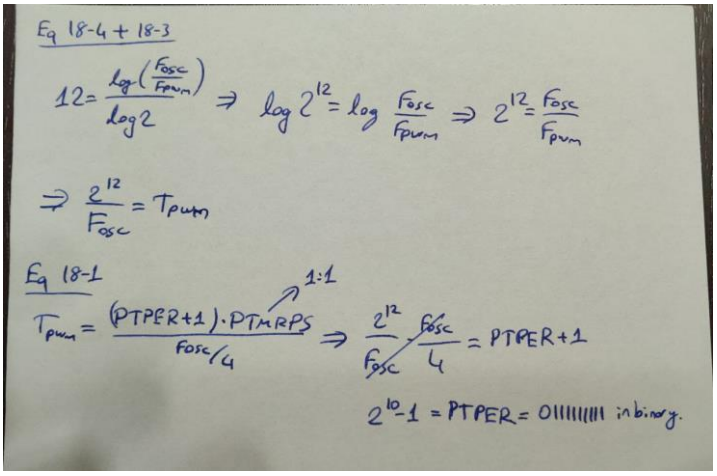
EQUATION 18-3: PWM FREQUENCY

$$\text{PWM Frequency} = \frac{1}{\text{PWM Period}}$$

EQUATION 18-4: PWM RESOLUTION

$$\text{Resolution} = \frac{\log\left(\frac{F_{OSC}}{F_{PWM}}\right)}{\log(2)}$$

- **PTPER: (PWM Period)** In order to calculate the PWM period we will use the below equations supplied in the data sheet page 185. This PIC supports up to 14 bits of PWM signal resolution, since we are going to be controlling motors with this signal our resolution will be limited by the gears of the motors. Still let's aim to use a 12 bit resolution in order to learn how this is calculated. We first use Equation 18-4. We put in resolution=12 and F_OSC=20Mhz since that is our crystal frequency and take PTMRPS (Prescaler) as 1:1. (This is instructed in the PTCON0 register) Then we find PTPER=2^10-1. PTPER=00000001 11111111. PTPERH=0b00000001, PTPERL=0b11111111;.



- **PDC: (PWM Duty Cycle)** PDC registers are our duty cycle values. Figure 18-12 and the PTMR register can be looked at to see how exactly they work. Basically speaking though since we have defined the PTPER value already. PDC has a range of 0 to PTPER for the values it can take. When PDC==PTPER we are outputting maximum power. PDC0 controls PWM0,1; PDC1 controls PWM2,3; PDC2 controls PWM4,5. Each PDC has lower and higher parts controlling it. In the below code I wrote (except the HIGH and LOW Byte Macros) we are simulating a pwm signal that is constantly changing. Please see Simulation 2: PWM.

```
/*
  PIC Proof of Concept for LED & PWM, for ME331 F2020.
  Simulated with PIC18F4431.
  PWM duty cycle gets smaller and smaller until it resets, LEDs blink opposite to each other.
  by: M. Çağatay Sipahioğlu
*/
//----------------------------------MACROS---------------------------------------
#define HIGH_BYTE(x) ((unsigned char)(((x)>>8)&0xFF))
#define LOW_BYTE(x) ((unsigned char)((x)&0xFF))

//-----------------------------------PINS-----------------------------------------
sfr sbit STBY_R_TRIS at TRISA2_bit;
sfr sbit STBY_L_TRIS at TRISA3_bit;

sfr sbit PWM_R_F_TRIS at TRISB0_bit;
sfr sbit PWM_R_B_TRIS at TRISB1_bit;
sfr sbit PWM_L_F_TRIS at TRISB2_bit;
sfr sbit PWM_L_B_TRIS at TRISB3_bit;
sfr sbit LED1_TRIS at TRISB4_bit;
sfr sbit LED2_TRIS at TRISB5_bit;

sfr sbit DIR_R_F_1_TRIS at TRISC0_bit;
sfr sbit DIR_R_F_2_TRIS at TRISC1_bit;
sfr sbit DIR_R_B_1_TRIS at TRISC2_bit;
sfr sbit DIR_R_B_2_TRIS at TRISC3_bit;

sfr sbit DIR_L_F_1_TRIS at TRISD0_bit;
sfr sbit DIR_L_F_2_TRIS at TRISD1_bit;
sfr sbit DIR_L_B_1_TRIS at TRISD2_bit;
sfr sbit DIR_L_B_2_TRIS at TRISD3_bit;



sfr sbit STBY_R at RA2_bit;          //Ports RA2-RA3 control motor standby.
sfr sbit STBY_L at RA3_bit;

sfr sbit PWM_R_F at RB0_bit;         //Ports RB0-RB1 control right front and back motor pwm.
sfr sbit PWM_R_B at RB1_bit;
sfr sbit PWM_L_F at RB2_bit;         //Ports RB2-RB3 control left front and back motor pwm.
sfr sbit PWM_L_B at RB3_bit;
sfr sbit LED1 at RB4_bit;           //Ports RB4-RB5 control the 2 LEDs.
sfr sbit LED2 at RB5_bit;

sfr sbit DIR_R_F_1 at RC0_bit;       //Ports RC0-1 control right front motor direction.
sfr sbit DIR_R_F_2 at RC1_bit;
sfr sbit DIR_R_B_1 at RC2_bit;       //Ports RC2-3 control right back motor direction.
sfr sbit DIR_R_B_2 at RC3_bit;

sfr sbit DIR_L_F_1 at RD0_bit;       //Ports RD0-1 control left front motor direction.
sfr sbit DIR_L_F_2 at RD1_bit;
sfr sbit DIR_L_B_1 at RD2_bit;       //Ports RD2-3 control left back motor direction.
sfr sbit DIR_L_B_2 at RD3_bit;
```

```
//-------------------------------------SETUP-------------------------------------
void setup_IO() {               //Setup_IO will assign input and output pins. (Output 0, Input 1)
    STBY_R_TRIS=0;              //OUTPUTS
    STBY_L_TRIS=0;
    PWM_R_F_TRIS=0;
    PWM_R_B_TRIS=0;
    PWM_L_F_TRIS=0;
    PWM_L_B_TRIS=0;
    LED1_TRIS=0;
    LED2_TRIS=0;
    DIR_R_F_1_TRIS=0;
    DIR_R_F_2_TRIS=0;
    DIR_R_B_1_TRIS=0;
    DIR_R_B_2_TRIS=0;
    DIR_L_F_1_TRIS=0;
    DIR_L_F_2_TRIS=0;
    DIR_L_B_1_TRIS=0;
    DIR_L_B_2_TRIS=0;}

void setup_Init() {                 //Setup_Init will initialize anything that requires initialization.
    PTCON0=0b00000000;
    PTCON1=0b10000000;
    PWMCON0=0b00111111;
    PWMCON1=0b00000001;
    PTMRH=0b00000000;
    PTMRL=0b00000000;
    PTPERH=0b00000001;
    PTPERL=0b11111111;
    ANSEL0=0x00;                //Analog multiplexes should be turned off otherwise there will arise problems.
    ANSEL1=0x00;
}

//-----------------------------------FUNCTIONS-----------------------------------
void pwm_demo(){
    int pwm=0;                  //Pwm demo by changing the pwm signal and seeing the output from the PROTEUS oscillator.
    STBY_R=1;                   //STBY should be high for the motors to work.
    STBY_L=1;                   //Set an initial direction for the motors.
    DIR_R_F_1=1;
    DIR_R_F_2=0;
    DIR_R_B_1=0;
    DIR_R_B_2=1;
    DIR_L_F_1=1;
    DIR_L_F_2=0;
    DIR_L_B_1=0;
    DIR_L_B_2=1;

   for(pwm=1;pwm<4096;pwm++){
      PDC0H=HIGH_BYTE(4096-pwm);
      PDC0L=LOW_BYTE(4096-pwm);
      PDC1H=HIGH_BYTE(4096-pwm);
      PDC1L=LOW_BYTE(4096-pwm);
      Delay_us(10);
   }
}

//-------------------------------------MAIN--------------------------------------
void main() {
    setup_IO();
    setup_Init();
    LED1=1; //Initialize the leds for the led demo, and simulate in the main loop.
    LED2=0;
    while(1){
        pwm_demo();
        LED1= !LED1;
        LED2= !LED2;}
}
```

9.5.4.3 SD Card, Temperature Sensor, LCD & Switch Code for PIC in MikroC (Note that the SD Card code doesn't work, needs further research.)

SD Cards can be written on using SPI communication with regard to the FAT file system. For this it is only logical to use libraries. The below code utilizes the MMC library of MikroC to implement a data writing operation on an appropriately connected SD Card. (See Circuitry)

While doing this application for PIC18F4431 MikroC gives the error of not enough RAM. Which is when I realized this PIC won't be able to utilize the SD Card communication, for this I have utilized SD Card communication in a **PROTEUS Simulation** using a similar PIC. (PIC18F4550) See, simulation 2:SD Card, LCD, Temperature.

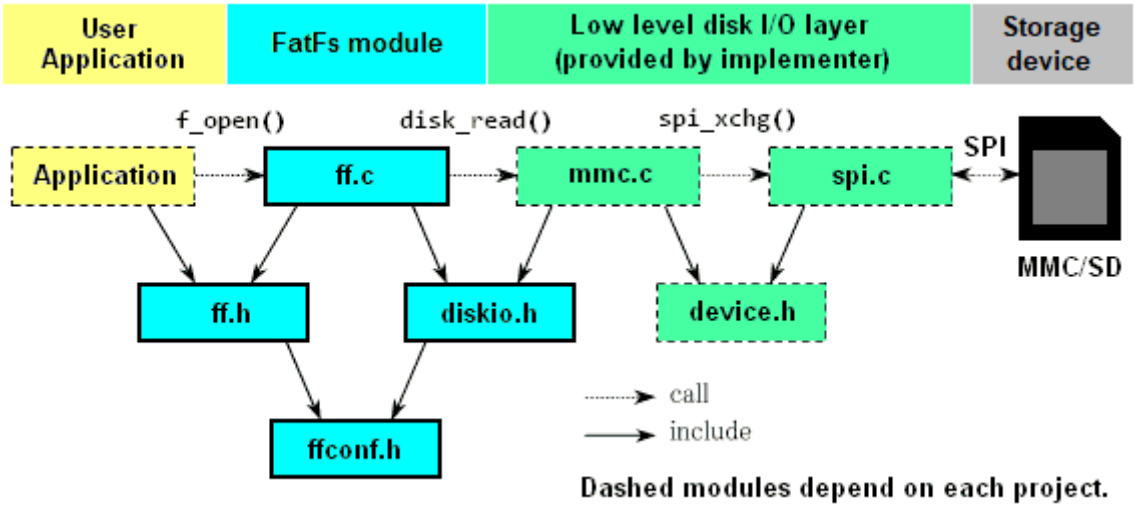
In order to simulate a SD Card in PROTEUS we must also create a digital SD image file on our computer and connect that digital image to the SD Card module. I have used the software **WinImage** to create that SD Card image in the FAT16 file system. Since the MMC library of MikroC supports FAT16. For the proteus demo we will take the temperature as input and show it on the LCD as well as writing it on the SD Card. (SD Card will be with the same code but on a different PIC)

Lastly when using a physical SD Card with the Arduino in the end I have used the software **MiniTool Partition Wizard** to format the SD Card. (Since it was used elsewhere and wasn't set to FAT file system) Before the initial format the SD Card wasn't even being seen by the PC when plugged in so I couldn't format it using the Windows itself. However after the format with this software it could now be seen by the computer. When tried with the Arduino in the debugging week the Arduino still couldn't see the SD Card so a full format was needed. For this I have this time used the "Overwrite format" function of the **SD Memory Card Formatter developed by the SD Association** and after this second complete format the SD Card could now both be read and written on by the PC and the Arduino.

A few notes on the code: This code and simulation (Simulation2) right now simulates LCD output, temperature ADC input and switch digital input. SD Card code is not working as of now. The first part of the code is just name declarations for our and libraries' use. There are two setup functions. The input-output setup function only declares the switch as an input. However because the switch is connected to a pin multiplexed with an analog input, setup_IO must be run after setup_Init in the main function. This is because inside the setup_Init function we are choosing which multiplexed pins will be analog and which will be digital and we can't declare a digital input at a multiplexed pin without first initializing the pin as digital. If the void_IO is run before void_Init the switch won't work.

In the main part of the code we are getting the temperature data and writing it on the LCD as well as the SD Card (SD not implemented). We are also using the switch as a kill switch for the LCD, when the switch gets pushed, the pin gets grounded and we read a 0. In this case the LCD turns off. However we also want the LCD to turn back on when we turn on the switch. It is not as straightforward as it sounds to achieve this however because the LCD resets when we turn it off and must be reinitialized to start working again. In order to achieve this I have declared two global variables i and j and initialized both to 0. When we turn off the switch I want to be able to count that and I want to count that as 1. In order to do this I have added 1 to i if the switch is turned off. But to stop the program from adding 1 to i as long as the switch is turned off I have put in a if statement where if i is bigger than j, the program won't add 1 to i even if the switch is still turned off. Now I have successfully counted the turn-off as 1 I must use this information to reinitialize the LCD when the switch is turned back on. When the switch is turned back on, the main loop goes straight to the temp_lcd function. At the beginning of this function I can now reinitialize the LCD if i is bigger than j. Since I have added 1 to i when the switch was turned off reinitialization happens and in order to reset the ground for the next time the switch is turned off I now

equate j to i. This way the next time switch is turned off I can do the same comparisons I did the first time.




```
/*
  PIC Proof of Concept for SD Card, LCD, Temperature Sensor, for ME331 F2020.
  Simulated with PIC18F4550.
  Temperature data is written both on the LCD and the SD Card.
  This program uses the following MikroC libraries: ADC, LCD, SPI, Mmc, Mmc_FAT16.
  It can be seen that this program uses about 1kB of RAM which the 18F4431 didn't have.
  by: M. Çağatay Sipahioğlu
*/


//------------------------------------PINS--------------------------------------
sfr sbit LCD_EN_Direction at TRISA1_bit;
sfr sbit LCD_RS_Direction at TRISA2_bit;         //LCD Pins must be named _Direction in order to work with the library.
sfr sbit SWITCH_TRIS at TRISA4_bit;
sfr sbit Mmc_Chip_Select_Direction at TRISA5_bit; //MMC_CS Pin must be named _Direction in order to work with the
library.

//sfr sbit SD_MISO_TRIS at TRISB0bit;
//sfr sbit SD_CLK_TRIS at TRISB1_bit;
//sfr sbit SD_MOSI_TRIS at TRISC7_bit;

sfr sbit LCD_D4_Direction at TRISD4_bit;
sfr sbit LCD_D5_Direction at TRISD5_bit;
sfr sbit LCD_D6_Direction at TRISD6_bit;
sfr sbit LCD_D7_Direction at TRISD7_bit;

//sfr sbit TEMP_TRIS at TRISE0_bit;    //Doesn't need to be defined thanks to ADC library.


sfr sbit LCD_EN at RA1_bit;          //LCD bits must be defined with these names for the library.
sfr sbit LCD_RS at RA2_bit;
sfr sbit SWITCH at RA4_bit;              //Debugging Switch
sfr sbit Mmc_Chip_Select at LATA5_bit;   //CS pin must be named like this in order to work with the library.

//sfr sbit SD_MISO at RB0_bit;
//sfr sbit SD_CLK at RB1_bit;            //Since there is only 1 SPI port. MISO, CLK and MOSI pins will be constant and
don't need defining.
//sfr sbit SD_MOSI at RC7_bit;

sfr sbit LCD_D4 at RD4_bit;          //Ports RD4-7 control the LCD along with RA0-RA1.
sfr sbit LCD_D5 at RD5_bit;
sfr sbit LCD_D6 at RD6_bit;
```

```
sfr sbit LCD_D7 at RD7_bit;

//sfr sbit TEMP at RE0_bit;            //Port RE0 connects the temperature sensor. (LM35) Doesn't need to be named, ADC library works straight with the Analog channel number.



 //--------------------------------VARIABLES-------------------------------------
unsigned int temp;
int i=0;   //Counters to understand if the SWITCH has been turned off.
int j=0;
int error;
char Temperature[] = "00";


//-----------------------------------SETUP-----------------------------------

void setup_IO() {                //Setup_IO will assign input and output pins. (Output 0, Input 1)
    SWITCH_TRIS=1;               //INPUTS
}

void setup_Init() {              //Setup_Init will initialize anything that requires initialization.
    ADCON0=0b00000001;           //ADC on for analog channel 0.
    ADCON1=0b00001110;           //VDD and GND are Analog reference. Only the 0th channel open. We should use the analog channels starting from AN0 since we can't turn for example AN5 on without turning AN0-4 on.
    ADCON2=0b10000010;           //Right justified, slowest.

     /*
    SPI1_Init();                 //Initialize SPI protocol
    SPI1_Init_Advanced(_SPI_MASTER_OSC_DIV64, _SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_LOW, _SPI_LOW_2_HIGH);
    Delay_us(10);
    error = Mmc_Init();
    MMC_Fat_Init();              //Initialize MMC Fat Systems
    SPI1_Init_Advanced(_SPI_MASTER_OSC_DIV4, _SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_LOW, _SPI_LOW_2_HIGH);
    Mmc_Fat_Assign("Data.TXT",0xA0);
       */
    ADC_Init();                  //Initialize Analog to Digital Conversion.
    Lcd_Init();                  //Initializes LCD pins & LCD.
    Lcd_Cmd(_LCD_CURSOR_OFF);    //Turns the Cursor off.
    Lcd_Cmd(_LCD_CLEAR);         //Clears the screen. (More commands can be looked up from the library.)
    Lcd_Out(1,1,"Hello");
    Delay_ms(1000);
}

//---------------------------------FUNCTIONS---------------------------------

void temp_lcd_demo(){
  if(i>j){
   Lcd_Init();       //When the SWITCH is off i goes 1 up. When the SWITCH goes back on this void initiates, LCD must
                       be reinitialized to work.
   Lcd_Cmd(_LCD_CURSOR_OFF);    //Reinitialize the LCD and equate i to j. When the SWITCH goes back off i can go up 1 more. And we can reinitialize again in this if.
   i=j;
  }

  temp = (ADC_Read(0))*500/1023;      //Read Temperature from ADC
  lcd_out(1, 1, "Temperature:");
  Temperature[0]='0'+ (temp/10)% 10;  //Tens of the temperature
  Temperature[1]='0'+ temp%10;  //Ones of the temperature
  Lcd_out(2,1, Temperature);
  Delay_ms(1000);
}
//-----------------------------------MAIN------------------------------------
void main() {
```

```
    setup_Init();
    setup_IO(); //IO setup must be done after initialization, since the proper digital/analog initialization of the ADCON
register is needed before assigning a digital input characteristic to the SWITCH. (Program doesn't work if setup_IO is
above setup_Init)
    while(1){
        if (SWITCH==0){         //We won't be using the SWITCH for debugging, rather we will demonstrate that we can
                                  use the SWITCH by turning off the LCD with it.
        Lcd_cmd(_LCD_TURN_OFF);
        if(i==j)  //Make i go 1 up when the SWITCH is off.
        i++;
        }
        else
        temp_lcd_demo();

        //sd_demo();
   }
}
```

9.5.4.4 MPU6050 Gyroscope for PIC in MikroC

There wasn't any official MPU6050 library for MikroC however there are community developed libraries for this gyroscope. Since I won't be able to simulate MPU6050 in PROTEUS due to the lack of libraries I haven't written a dedicated code for it. However MPU6050 outputs an analog output and the community libraries takes this output and converts it to the pitch, yaw and roll values. Therefore MPU6050 can be treated like the analog temperature sensor above since they both utilize the ADC library. And the proof of concept of the temperature sensor should show that we can manipulate analog data, given the communication libraries.

9.6 Tools Used

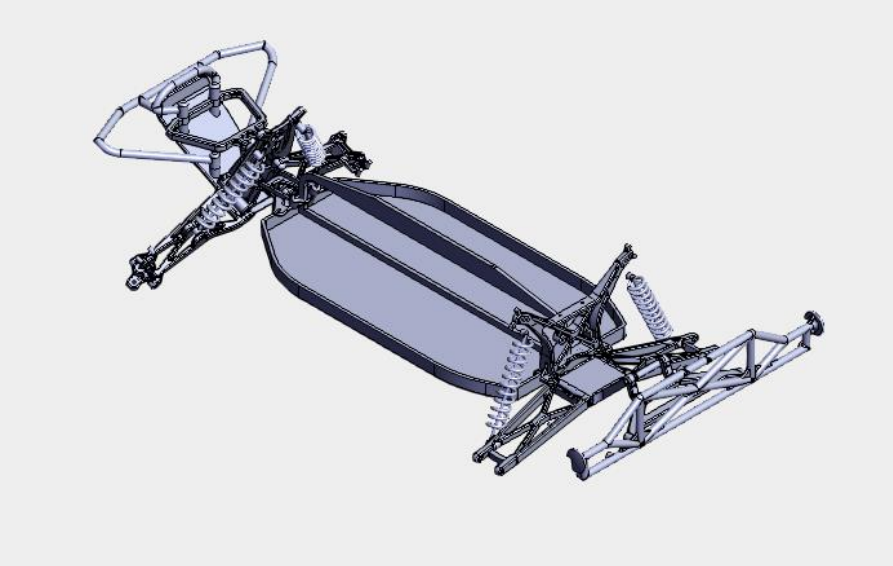
- **Proteus**, for simulating the PIC circuits.
- **MikroC PRO for PIC**, for PIC coding.
- **WinImage**, for creating a virtual SD Card image for the Proteus simulation.
- **MiniTool Partition Wizard, SD Memory Card Formatter** for formatting the physical SD Card.
- **Fritzing**, for the Arduino schematic.

9.7. Attachments

- PIC Catalogue.xlsx
- Datasheets for
    - PIC18F4431
    - PIC18F4550
    - TB6612FNG, Motor Driver
    - L7805CV, Voltage Regulator
- Proteus Simulations (And the Simulation Videos) with MikroC codes for
    - Simulation1 - LED & PWM
    - Simulation2 - SD-LCD-TEMP-SWITCH
- The Full Circuit in Proteus

10. Structural Design

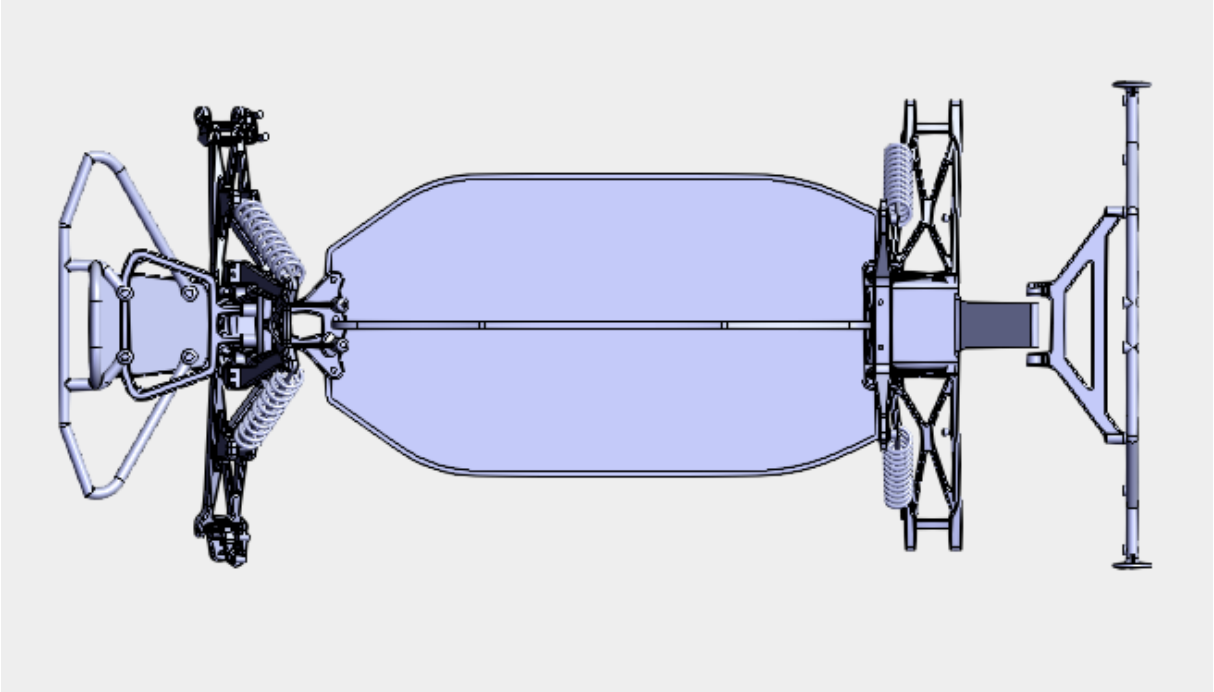
The structure of the robot was designed based on a remote controlled car structure. The relevant parts of an RC car were kept to form a base structure.



There is a 27,5 cm x 15,0 cm platform present in the middle of the chassis to provide sufficient space for circuit components and batteries to be placed. The layout of the circuit is to be arranged in order to prevent any electrical and mechanical conflicts between circuit components and chassis members.

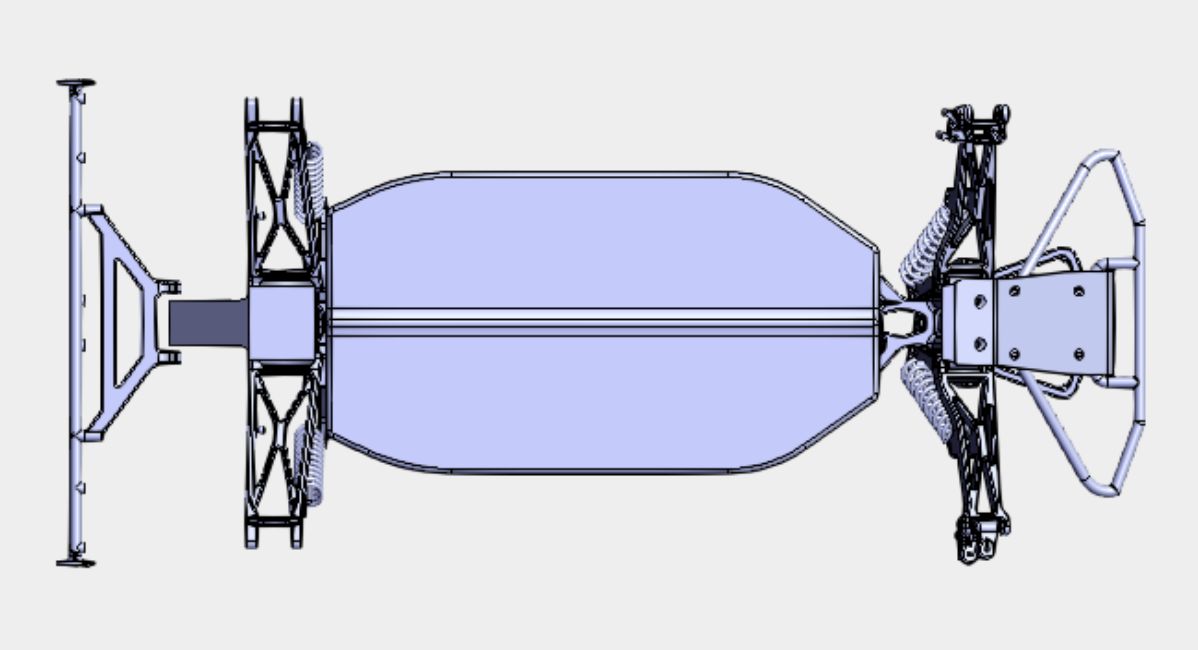
Directional Views of the Structure

Top View of the Chassis:

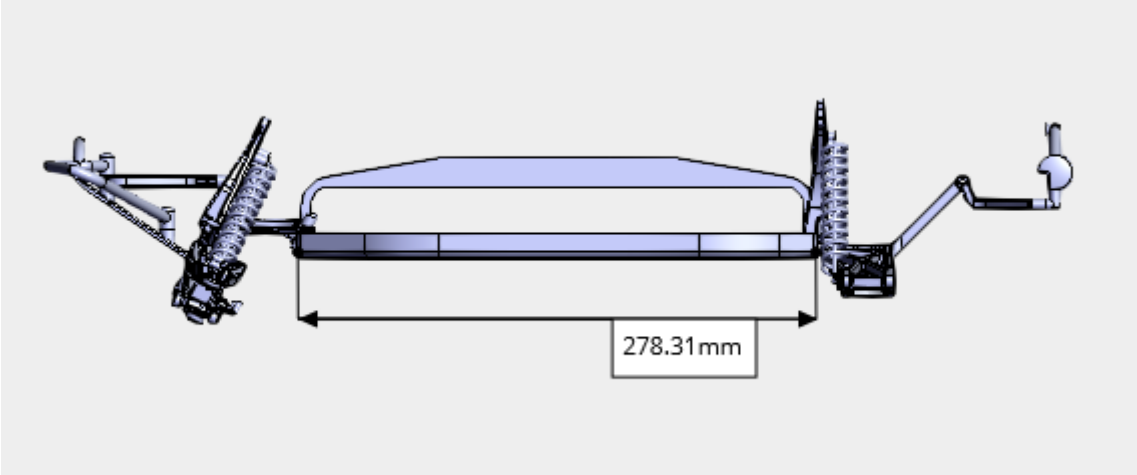


The 27,5 cm x 15,0 cm platform can be seen in the middle. There are front and back end bumpers present in the design to absorb any direct damage to the main platform.

Bottom View of the Chassis:

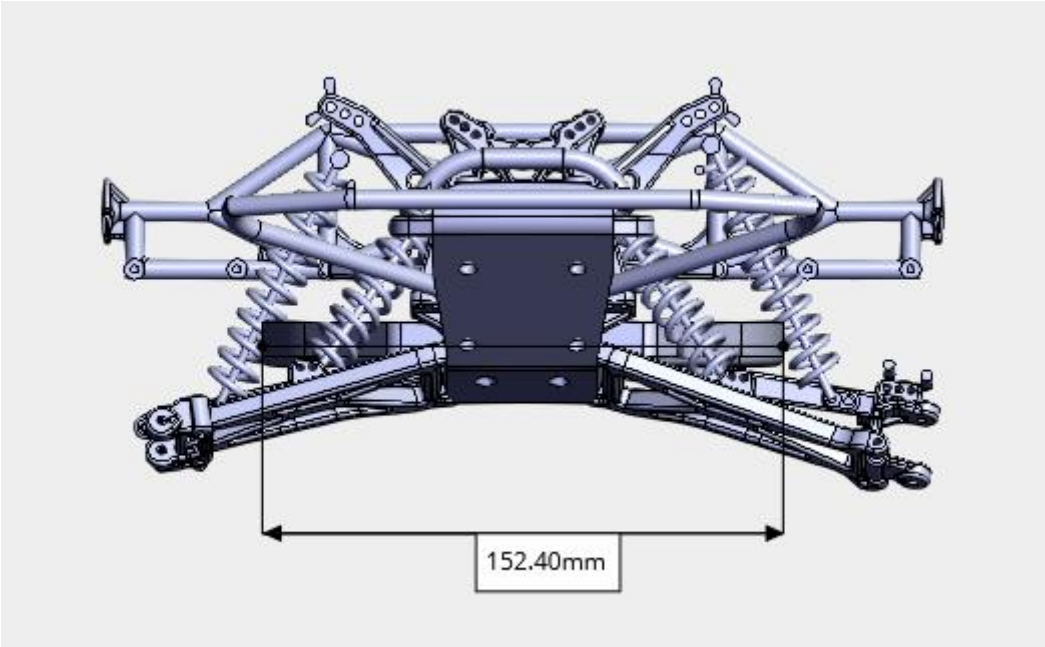


Left View of the Chassis:



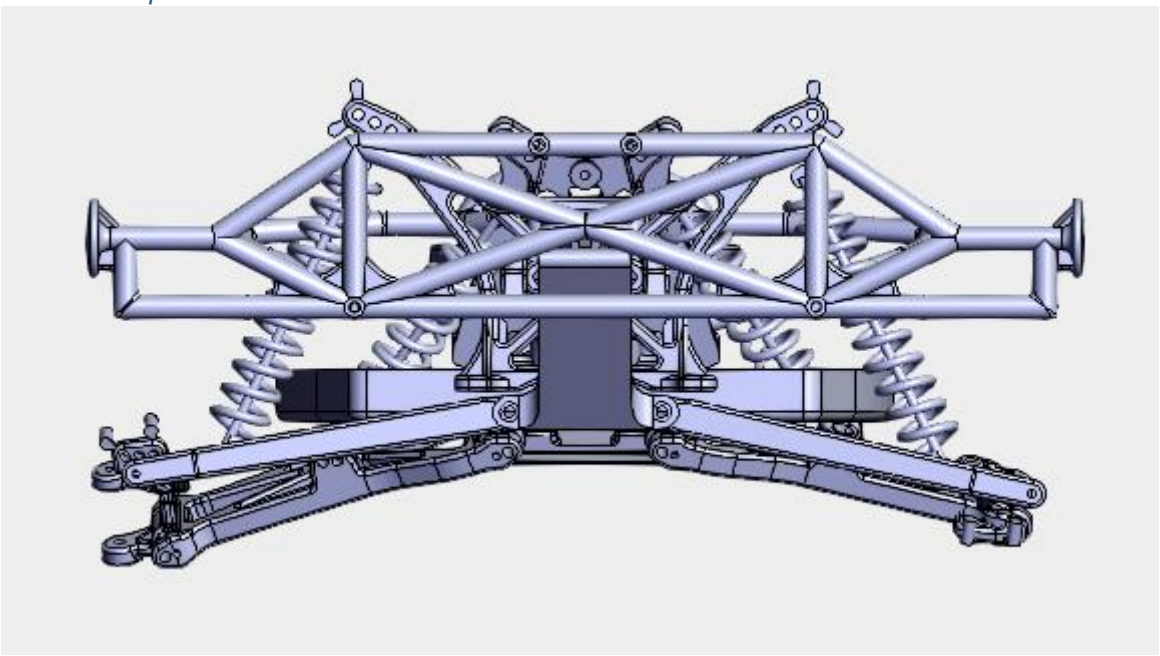
An additional support beam on the platform is present to prevent buckling in the event of a hard crash.

Front View of the Chassis:



There are shock absorbers present on every wheel to prevent extreme vibrations that may occur while the robot is moving in the farming fields. Also in order to minimize vibration effect on the structure and circuit components, the velocity of the robot in its path is optimized.

Back View of the Chassis:



11. Prototype Production

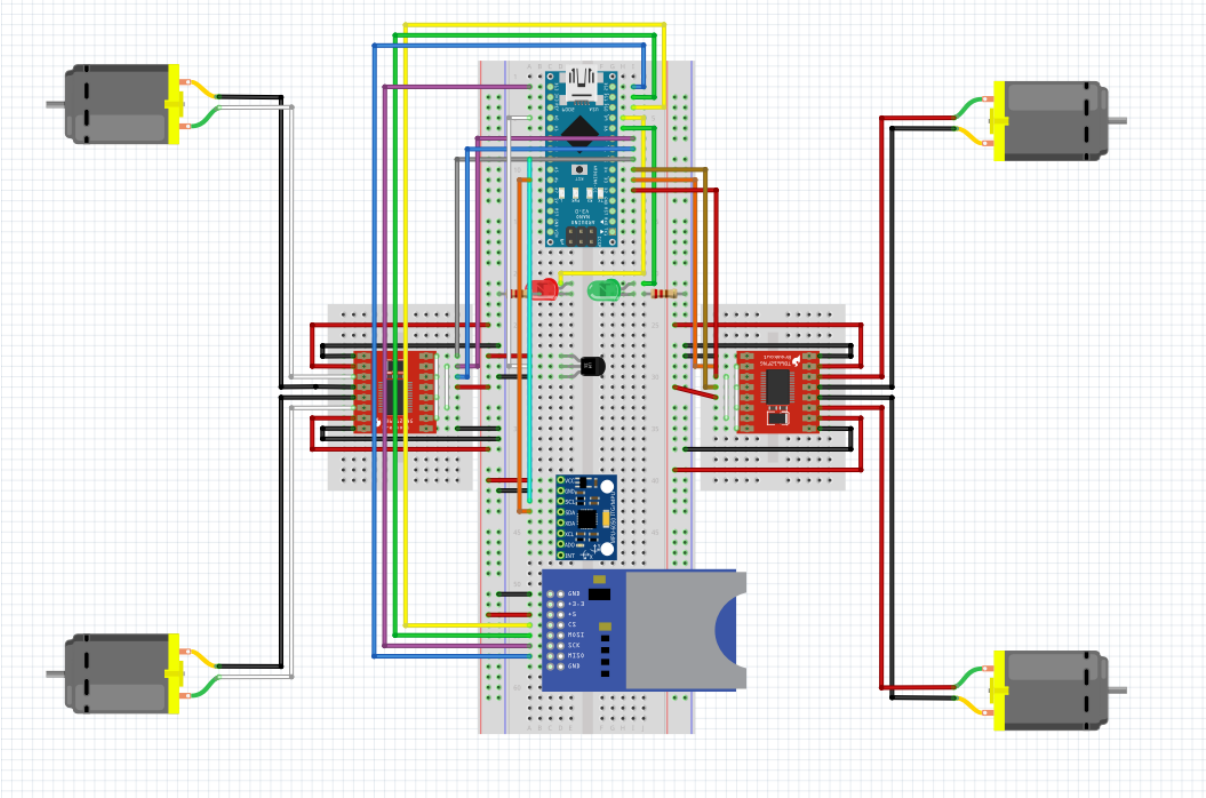
by: M. Çağatay Sipahioğlu

I have built the physical prototype since I was already building the circuitry physically.

11.1 Physical Circuitry

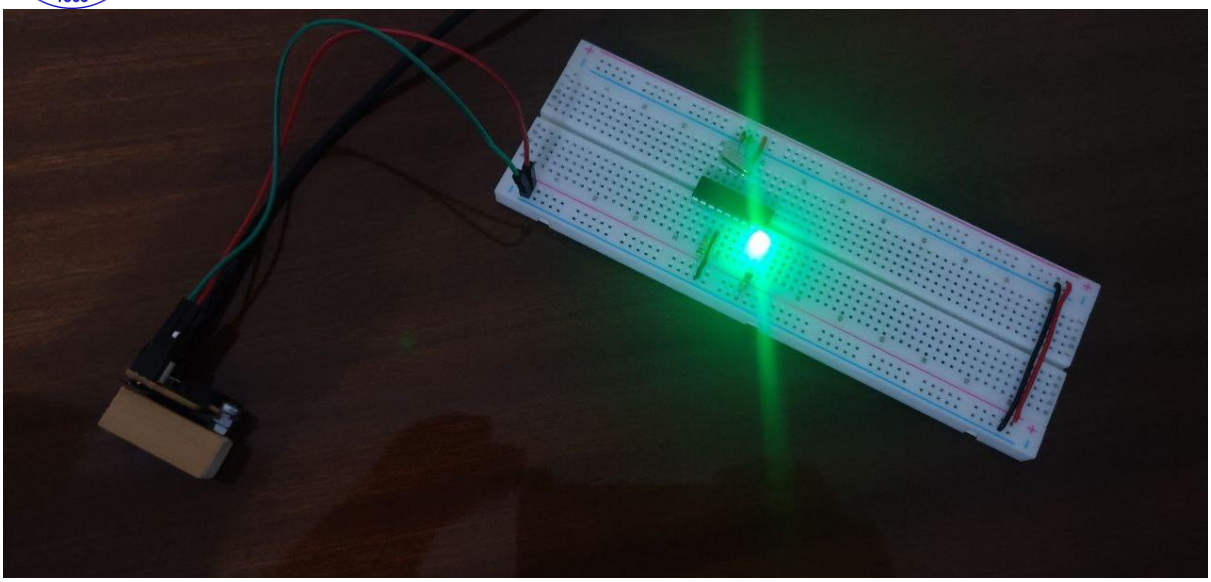
Arduino

Circuitry is the same for the Arduino however some parts of the circuitry from the PIC can be negated thanks to the on board modules of the Arduino itself. Namely: MCLR circuit, Oscillator circuit, ICSP circuit, Voltage Regulator Circuit. And the motor drivers are used as ready modules. (That's how they were bought) Also we used Arduino Nano over Uno since it takes much smaller space, can be used with breadboards and offers the same amount of pins as the UNO. Arduino Circuit is like the schematic below: (Tool used: **Fritzing)**

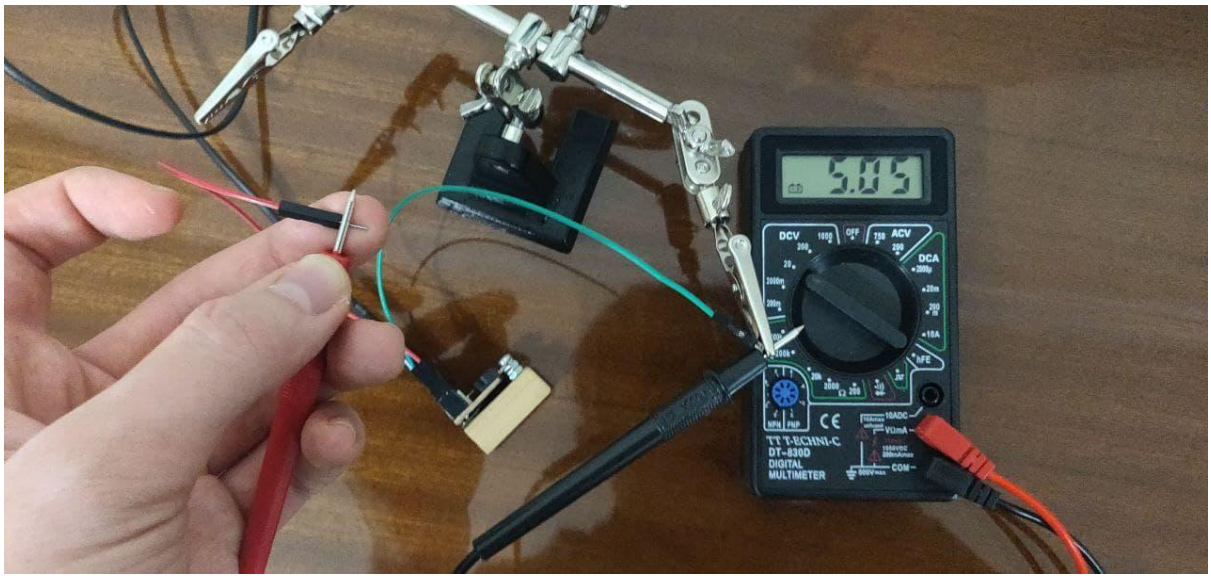


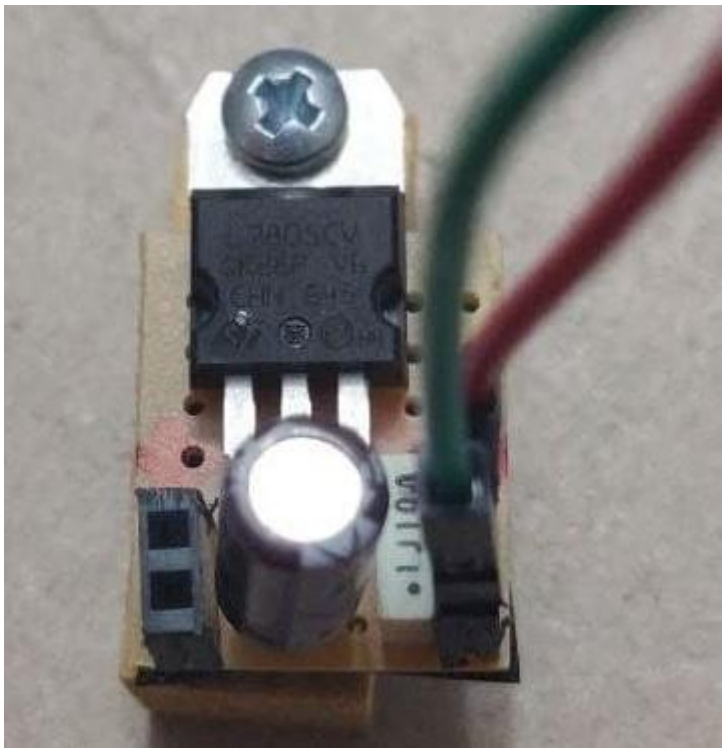
Early PIC Circuitry Used Before Pivoting to Arduino

In order to power the PIC for prototyping in the early days, I have cut off the end of an old Nokia adapter and soldered jumper heads to the positive and negative terminals. The adapter was supposed to give out +5V which I could use for PIC powering but it gave 7.5V instead, probably because it is a very old piece of electronics. So I built the voltage regulator circuit I have drawn in the circuitry part on a piece of electronics stripboard and soldered the connections in the right order along with 2 sets of female headers so I could draw power with jumpers. I then screwed the regulator on to a piece of small wood so I could use the screw as a heatsink. I have successfully built a LED blink example with a PIC16F628A. (PIC18F4431 hadn't arrived back then) But it was very time consuming to test code and circuits physically every time so for the rest of the project I have used PROTEUS to test the code and circuitry. Below are the photos of the aforementioned modules:




7.6V


5.05
DCV
ACV
DCA
OFF
DIGITAL
MULTIMETER

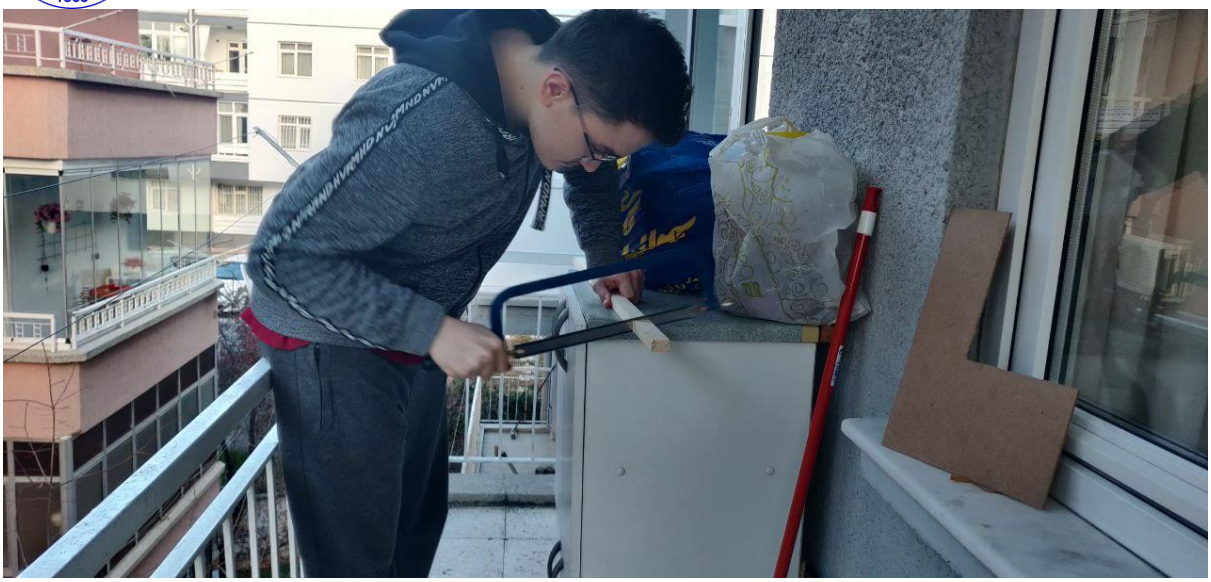


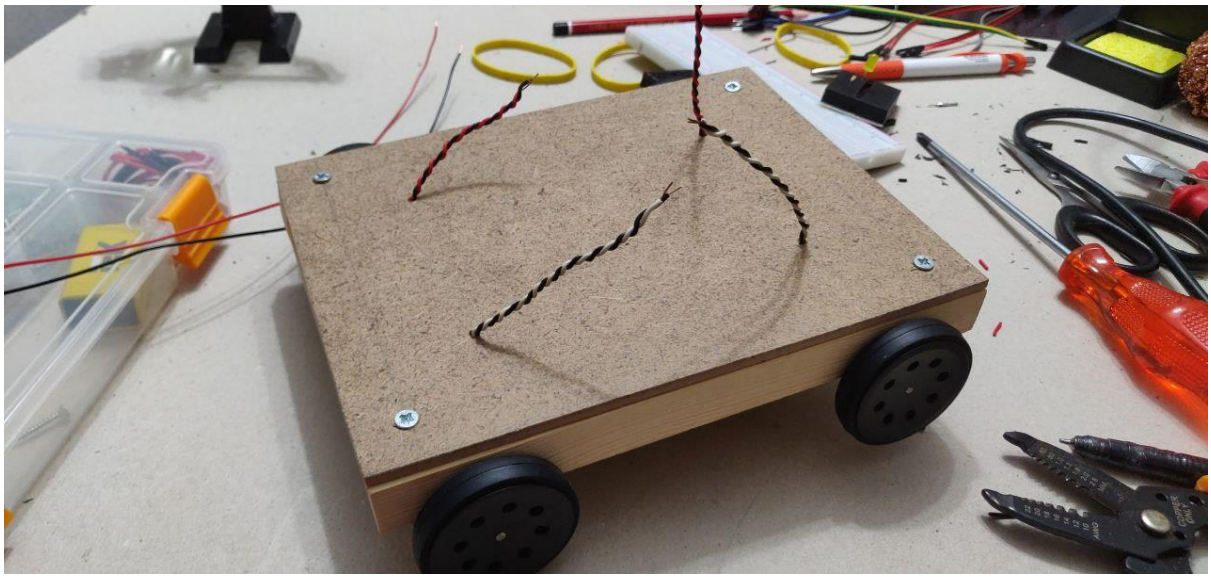
11.2 Woodworking

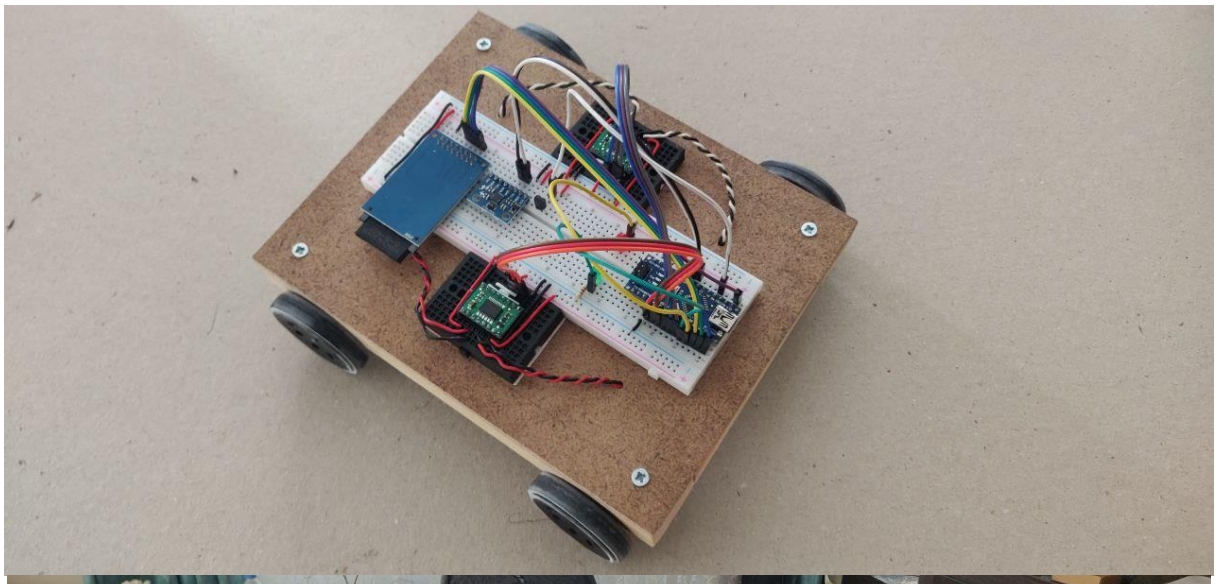
For the prototype we couldn't access the chassis suggested by the Structural Design. Therefore I have bought some rectangular wooden beams, (19x27mm cross section) some plastic corner pieces that can be screwed in, appropriate length screws and a plate of plywood out of which I will cut a plate to be the platform the breadboard and the modules will be mounted on. I first cut the wood pieces to their appropriate lengths. I aimed for a chassis that is 20x15cm. So I cut 2 20 cm and 2 10 cm pieces. With the addition of the width of the beams the total came up to a chassis that is 20x15.5 cm. I have removed some material while sanding the edges and the faces.

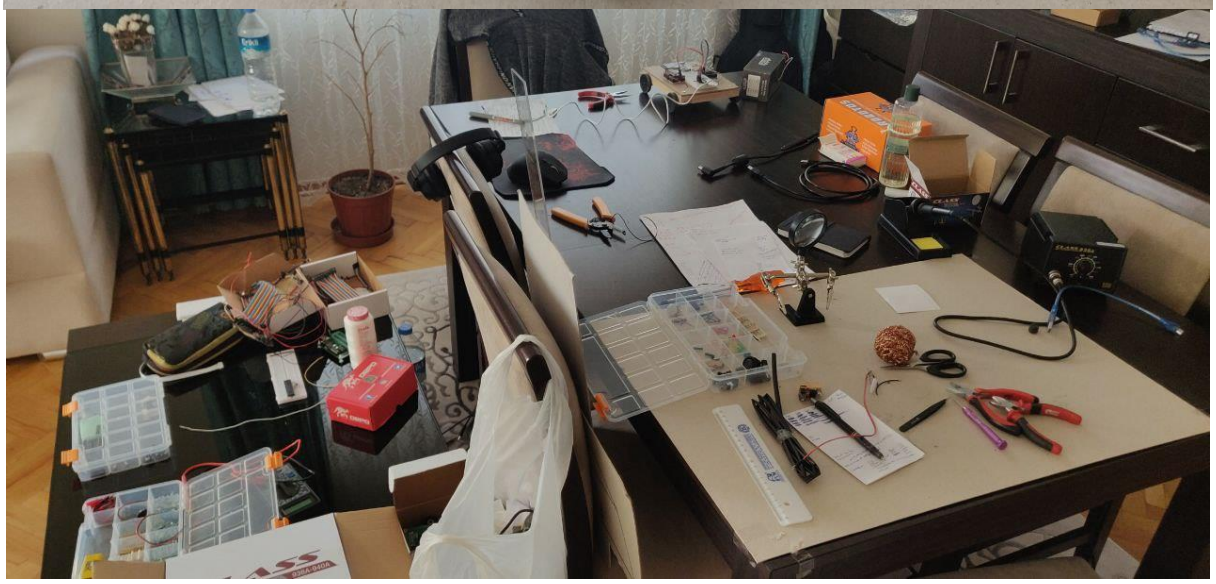
I then put the beams together, screwed in the corner pieces from the inside and I screwed the plywood plate on top. I have then drilled some small pilot holes for the motor bracket and screwed the motors with their brackets into the bottom of the chassis at equal distances. (Or as best as I could) I have also mounted the tires on to the motor shafts.

I then proceeded to drill some small holes to take the motor wires through the plywood to the breadboards above. And I routed the wires through the holes. I then fixed the breadboard circuitry to the plywood and finished the connections. The finished robot totalled at 500 grams. (Production version would have a lighter chassis since the wood I used was somewhat dense, but this extra weight can be thought of as the weight of the battery.)









Production Environment

12. Conclusion

In this report, the project planning, design and realization processes of an agricultural robot project for Bogazici University ME 331: Mechatronics Course were introduced. Processes were realized using engineering tools and softwares.

As a result of the project, a prototype that will serve as a technology demonstrator and a possible sustainable solution for agricultural activities in Turkey was designed.

Electronic Components Used

MCU: PIC18F4431

Motors: ROBOTUS 6V 180RPM DC Motor

Motor Driver: TB6612FNG

Voltage Regulator: L7805CV

Prototype Dimensions

The body of the prototype : 15,5 cm x 20,0 cm in the shape of a rectangular box

Wheels: 50 mm diameter rubber tires

13. References

[5.1] THE 17 GOALS | sustainable development. (n.d.). Retrieved February 11, 2021, from https://sdgs.un.org/goals

[5.2] Press corner. (n.d.). Retrieved February 11, 2021, from https://ec.europa.eu/commission/presscorner/detail/e%20n/ip_19_6691

[5.3] The Sustainable Development Goals Report - 2020. (n.d.). Retrieved February 11, 2021, from https://unstats.un.org/sdgs/report/2020/

[5.4] What is sustainable development? – United Nations Sustainable Development. (n.d.). Retrieved February 11, 2021, from https://www.un.org/sustainabledevelopment/blog/2015/09/what-is-sustainable-development

[6.1] Collins, D. (n.d.). FAQ: What's the difference between TORQUE CONSTANT, back emf constant, and Motor constant? Retrieved February 11, 2021, from https://www.motioncontroltips.com/faq-difference-between-torque-back-emf-motor-constant

[6.2] Fxsolver. (n.d.). Retrieved February 11, 2021, from https://www.fxsolver.com/

[7.3] Zucconi, A. (2020, December 23). The secrets of Colour interpolation. Retrieved February 14, 2021, from https://www.alanzucconi.com/2016/01/06/colour-interpolation/

[7.3] https://stackoverflow.com/questions/7530627/hcl-color-to-rgb-and-backward

[7.3] https://github.com/hsluv/hsluv-java

[8] Language reference. (n.d.). Retrieved February 14, 2021, from https://www.arduino.cc/reference/en/

[8] George, L. (2017, December 26). Led blinking with pic microcontroller - mplab xc8 compiler. Retrieved February 14, 2021, from https://electrosome.com/led-pic-microcontroller-mplab-xc8/

[8] Basics of spi. (2020, May 11). Retrieved February 14, 2021, from https://openlabpro.com/guide/basics-of-spi/

[8] SD card read/write USING PIC18F4550. (2020, May 11). Retrieved February 14, 2021, from https://openlabpro.com/guide/raw-sd-readwrite-using-pic-18f4550/

[8] Bindu, B. (2019, December 19). Reading/Writing to sd card with PIC USING MCC. Retrieved February 14, 2021, from https://www.studentcompanion.co.za/reading-and-writing-to-sd-card-with-pic-microcontroller-using-mplab-code-configurator/

[8] https://stackoverflow.com/questions/35486045/arduino-reading-sd-file-line-by-line-c

[8] https://forum.arduino.cc/index.php?topic=580261.0

[8] https://forum.arduino.cc/index.php?topic=246654.0

[8] https://github.com/pozyxLabs/Pozyx-Arduino-library/issues/61

[8] https://forum.arduino.cc/index.php?topic=165403.0

[8] https://arduino.stackexchange.com/questions/14125/arduino-digitalwrite-1-or-0-instead-of-high-or-low

[8] https://raw.githubusercontent.com/jarzebski/Arduino-MPU6050/master/MPU6050_gyro_pitch_roll_yaw/MPU6050_gyro_pitch_roll_yaw.ino

[8] https://forum.arduino.cc/index.php?topic=506775.0

[8] https://arduino.stackexchange.com/questions/65888/arduino-sd-card-open-file-modes-append-overwrite

[8] MPU 6050 Tutorial: How to Program Mpu 6050 with arduino. (n.d.). Retrieved February 14, 2021, from https://create.arduino.cc/projecthub/MissionCritical/mpu-6050-tutorial-how-to-program-mpu-6050-with-arduino-aee39a

[8] Dejan. (2021, February 05). Arduino and MPU6050 accelerometer and GYROSCOPE TUTORIAL. Retrieved February 14, 2021, from https://howtomechatronics.com/tutorials/arduino/arduino-and-mpu6050-accelerometer-and-gyroscope-tutorial/

[8] https://arduino.stackexchange.com/questions/75184/sd-card-with-arduino-uno?rq=1

[8] https://arduino.stackexchange.com/questions/17828/how-to-use-a-microsd-card-reader-with-sck-instead-of-clk?rq=1

[8] https://arduino.stackexchange.com/questions/17062/sd-card-fails-to-initialize?noredirect=1&lq=1

[8] https://arduino.stackexchange.com/questions/33455/why-is-sd-card-failing-to-begin?noredirect=1&lq=1

[8] Using the SD library to log data. (n.d.). Retrieved February 14, 2021, from https://www.arduino.cc/en/Tutorial/LibraryExamples/Datalogger

[8] Using the SD library to retrieve information over a serial port. (n.d.). Retrieved February 14, 2021, from https://www.arduino.cc/en/Tutorial/LibraryExamples/CardInfo

[8] SD card module with Arduino: How to read/write data. (n.d.). Retrieved February 14, 2021, from https://create.arduino.cc/projecthub/electropeak/sd-card-module-with-arduino-how-to-read-write-data-37f390

[8] https://github.com/arduino/Arduino/wiki/Unofficial-list-of-3rd-party-boards-support-urls

[8] Guindon, C. (n.d.). Program your arduino like a pro with the eclipse c/c++ ide. Retrieved February 14, 2021, from https://www.eclipse.org/community/eclipse_newsletter/2017/april/article4.php

[8] Staff, M. (2019, February 21). How to program arduino with eclipse. Retrieved February 14, 2021, from https://www.digikey.com/en/maker/blogs/2018/how-to-program-arduino-with-eclipse

[8] Kibalo, T. (2014, May 05). Execute open-source arduino code in a pic microcontroller using the mplab ide. Retrieved February 14, 2021, from https://circuitcellar.com/cc-blog/execute-open-source-arduino-code-in-a-pic-microcontroller-using-the-mplab-ide/

[8] https://www.microchip.com/forums/m1052685.aspx

[8] Lloyd, J. (2020, April 09). How to write pseudocode. Retrieved February 14, 2021, from https://www.wikihow.com/Write-Pseudocode

[8] XC8 İle Microchip PIC – Giriş. (n.d.). Retrieved February 14, 2021, from https://www.mcu-turkey.com/xc8-ile-microchip-pic-giris/

[8] XC8 İle Microchip PIC – I/O ve Register Kullanımı. (n.d.). Retrieved February 14, 2021, from https://www.mcu-turkey.com/xc8-ile-microchip-pic-io-ve-register-kullanimi/

[8] Nedir Bu Xc8? (n.d.). Retrieved February 14, 2021, from https://www.picproje.org/index.php?topic=52980.0

[8] Excel data LOGGER using PIC16F877A and Dht11 sensor. (2018, April 19). Retrieved February 14, 2021, from https://simple-circuit.com/pic16f877a-excel-plx-daq-data-logger/

[8] https://www.microchip.com/forums/m/tm.aspx?m=859647&p=1

[8] https://www.microchip.com/forums/m/tm.aspx?m=1042878&p=1

[8] https://www.microchip.com/forums/m1082058.aspx

[8] Programming Microchip PIC16F USING mcc (volume 2). (n.d.). Retrieved February 14, 2021, from https://developerhelptraining.thinkific.com/courses/MCC7to11

[8] https://www.microchip.com/forums/m1052685.aspx

[8] PIC projects with Ccs C compiler. (2019, May 31). Retrieved February 14, 2021, from https://simple-circuit.com/ccs-c-projects/

[9.2]

- PIC Selection
  https://www.microchip.com/ParamChartSearch/Chart.aspx?branchID=1005
  https://www.youtube.com/watch?v=R7lPEeUyNNA&ab_channel=MicrochipTechnology
- 8vs16vs32bit
  https://electronics.stackexchange.com/questions/106933/how-different-are-8-bit-microcontrollers-from-32-bit-microcontrollers-when-it-co#:~:text=In%20general%2C%20going%20from%208,doing%20arithmetic%20and%20logical%20operations.&text=But%20some%208%2Dbit%20micros,96%20bytes%20on%20a%20PIC16).
  https://www.microchip.com/forums/m774614.aspx
  https://www.youtube.com/watch?v=_SkpnG571z8&ab_channel=ArrowElectronics
- PIC vs AVR
  https://circuitdigest.com/article/comparison-and-difference-between-pic-vs-avr-microcontroller-architecture#:~:text=Latest%20version%20of%20both%20PIC,AVR%20is%20very%20much%20same.&text=Whereas%20AVR%20has%20integrated%20fuse,can%20be%20set%20in%20code.
- EEPROM
  https://www.electro-tech-online.com/threads/how-is-hef-eeprom-are-microchip-doomed.141614/
- UART
  https://electrosome.com/uart-pic-microcontroller-mplab-xc8/#:~:text=UART%20stands%20for%20Universal%20Asynchronous,devices%20should%20be%20made%20common.
- SENT
  https://en.wikipedia.org/wiki/SENT_(protocol)
- SPI
  https://circuitdigest.com/microcontroller-projects/pic16f877a-spi-communication-tutorial#:~:text=The%20term%20SPI%20stands%20for,Display%20controllers%20and%20much%20more.
- I2C, I2S, CAN
  https://circuitdigest.com/microcontroller-projects/i2c-communication-with-pic-microcontroller-pic16f877a
  https://en.wikipedia.org/wiki/I%C2%B2S

https://www.csselectronics.com/screen/page/simple-intro-to-can-bus/language/en

- Modules
  http://picguides.com/beginner/digital.php#:~:text=The%20name%20TRIS%20is%20a,that%20pin%20is%20an%20input.
  https://www.instructables.com/Complete-Motor-Guide-for-Robotics/
  https://medium.com/husarion-blog/10-steps-to-choosing-the-right-motors-for-your-robotic-project-bf5c4b997407
  https://www.servomagazine.com/magazine/article/tips_for_selecting_dc_motors_for_your_mobile_robot
  https://www.pololu.com/docs/0J21/5.c
  https://learn.sparkfun.com/tutorials/tb6612fng-hookup-guide/all
  https://www.youtube.com/watch?v=mRxx10HaMp4&ab_channel=Elektro%C4%B0nceleme
  https://arduino.stackexchange.com/questions/37075/correct-capacitors-to-use-with-l7805-voltage-regulator
  https://www.engineersgarage.com/tech-articles/choosing-battery-for-robots/

[9.3]

- https://circuitdigest.com/microcontroller-projects/gps-interfacing-with-pic16f877a
- http://www.studentcompanion.co.za/pic-microcontroller-communication-with-spi-bus-xc8/
- https://medium.com/jungletronics/pic-18-spi-intro-to-spi-protocol-a2687eba2fd7
- https://www.youtube.com/watch?v=PGXWm-BNPd0&ab_channel=SimpleProjects
- https://www.studentcompanion.co.za/interfacing-sd-card-with-pic-microcontroller-mikroc/
- http://www.ccsinfo.com/forum/viewtopic.php?t=40126
- https://www.microchip.com/forums/m211445.aspx

[9.4]

- https://www.youtube.com/watch?v=TfLgg1m2Mwo&ab_channel=elektrikbilim
- https://youtube.com/playlist?list=PLX6WBJrzZfFM49ViJRr3zyRcNrL3pO917
- https://www.microchip.com/wwwAppNotes/AppNotes.aspx?appnote=en012145
- https://www.electronicwings.com/pic/dc-motor-interfacing-with-pic18f4550
- https://www.instructables.com/Understanding-ICSP-for-PIC-Microcontrollers/
- http://www.learningaboutelectronics.com/Articles/How-to-connect-an-ICSP-interface.php
- https://deepbluembedded.com/mpu6050-with-microchip-pic-accelerometer-gyroscope-interfacing-with-pic/
- https://microcontrolandos.blogspot.com/2013/12/pic-mpu6050.html
- https://www.electronicwings.com/pic/mpu6050-gyroscope-accelerometer-temperature-interface-with-pic18f4550

[9.5.1]

- https://www.youtube.com/watch?v=RDdpByMeOPI&ab_channel=LuisAlbertoBorjaG%C3%B3mez
- https://www.youtube.com/watch?v=YwOcBT2aHjo&ab_channel=LuisAlbertoBorjaG%C3%B3mez

[9.5.2]

- https://www.microchip.com/en-us/development-tools-tools-and-software/mplab-xc-compilers
- https://www.microchip.com/devtoolthirdparty/CompanyListing.aspx?compid=7ab4cf53-fce0-4e5c-aa87-250891e5a198
- https://www.youtube.com/watch?v=adokiy48zLQ&ab_channel=ProjeAkademisi
- https://www.youtube.com/watch?v=u9q2DMWBKRA&ab_channel=ProjeAkademisi
- https://www.youtube.com/watch?v=EYRdg6AjPy8&ab_channel=ProjeAkademisi
- https://www.youtube.com/watch?v=yG3SUK-tG_g&ab_channel=ProjeAkademisi

- https://www.youtube.com/watch?v=OdXS87rz4b8&ab_channel=ProjeAkademisi
- https://www.youtube.com/watch?v=ly2hqaMT4SU&ab_channel=ProjeAkademisi

[9.5.3]

- https://www.youtube.com/watch?v=1Tfz1JfQjDI&ab_channel=DaveFisher
- https://www.youtube.com/watch?v=Xdo-BZpFf7c&ab_channel=DaveFisher
- https://www.youtube.com/watch?v=6WoY_J78sbs&ab_channel=DaveFisher
- https://www.youtube.com/watch?v=6WoY_J78sbs&ab_channel=DaveFisher
- https://www.youtube.com/watch?v=ynD8dSJFZPg&ab_channel=DaveFisher
- https://www.microchip.com/forums/m164179.aspx
- https://www.embedded.com/introduction-to-watchdog-timers/
- https://www.embedded.com/introduction-to-watchdog-timers/
- https://www.youtube.com/watch?v=x5PZDOp-148&ab_channel=SaravananAL
- https://www.pic18f.com/uncategorized/2010/07/06/tutorial-5-ad-conversion-in-c/#more-116

[9.5.4]

- https://www.youtube.com/watch?v=Zjv3fNCcVsI&ab_channel=BinderTronics
- https://www.youtube.com/watch?v=Tiw41qHowr0&ab_channel=StudentCompanion
- https://www.youtube.com/watch?v=bj1bSwsqdmc&ab_channel=%C4%B0slam%C3%87%C3%B6zel
- https://www.youtube.com/watch?v=oaOgcYOC2FM&ab_channel=S%C3%BCleyman%C5%9EEKER
- https://www.microchip.com/forums/m558033.aspx
- https://www.youtube.com/watch?v=Lw-fhR-vmxo&ab_channel=DaveFisher
- https://download.mikroe.com/documents/compilers/mikroc/pic/help/mmc_library.htm
- SD Card
  http://www.studentcompanion.co.za/pic-microcontroller-communication-with-spi-bus-xc8/
  https://www.studentcompanion.co.za/forums/topic/how-to-use-fatfs-library-in-mplab-xc8-without-mplab-code-configurater/
  https://www.microchip.com/forums/m1036310.aspx
  https://www.microchip.com/forums/m1035062.aspx
  http://www.piclist.com/images/ORG/elm-chan/http/docs/mmc/mmc_e.html
  https://www.microchip.com/SWLibraryWeb/product.aspx?product=Memory%20Disk%20Drive%20File%20System
  https://www.studentcompanion.co.za/interfacing-sd-card-with-pic-microcontroller-xc8/
  https://www.picproje.org/index.php?topic=54274.0
  https://www.studentcompanion.co.za/interfacing-sd-card-with-pic-microcontroller-xc8/
  https://simple-circuit.com/pic18f46k22-sd-card-fat32-mikroc/
  https://simple-circuit.com/mikroc-dht22-data-logger-sd-card/
- Not enough RAM error
  https://forum.mikroe.com/viewtopic.php?f=88&t=51613

14. Appendices

14.1 Appendix 1 - Devlog

15.12.2020

- Brainstorming and researching project ideas. Results:
    - Industrial Arm
    - Agricultural Robot
    - Swarm Robotics
    - 3D Printer/Writer
    - Drone

22.12.2020

- Determine the scope of the project:
    - Mobile robot that will traverse farming terrain.
    - Keeps track of location data, takes measurements (Water, pH, temperature)and saves them.
    - Path find or map when necessary.
- Possible Improvements:
    - Dry spots can be watered by an automatic irrigator.
    - pH can be balanced by necessary materials with a different robot or an addition to the existing robot.
- Workload distribution
    - Mapping Software: Çağrı: Determine the shape and dimensions of farmland. Calculate a path, give necessary movement information to the robot.
    - Modelling: Yusuf: Transfer functions, dynamic analysis.
    - Structural Design: Mehmet: Chassis, wheels, and tyres, load calculations.
    - Control Software: Cem
    - Electronics: Çağatay
    - Environment Simulation when needed.

26.12.2020

- Software (Computer)
- Software (Embedded)
    - Position, speed, motor control code
- Electronics

    - PIC, SD card module, IR module, Gyro module?, motor drivers, battery specifications?, GPS module?
- Modelling
    - Torque needed, suitable motor. Suspension as further improvements after project finish.
- Structural
- Simulator

29.12.2020

- Modelling
    - Wheel on grounds modelling
    - Proper motor choice
- Electronic
    - Battery choice
    - PIC choice, narrowing down.
- Embedded Programming
    - Motor control
    - SD Card read, write
- Computer Software
    - Any farmland can be divided into rectangles and be traversed as such.

05.01.2021

- Electronics
    - Done
        - Use PIC18F4431, has 6pwm, spi interface and is available in Turkey.
        - Use motor driver TB6612FNG, more efficient than classic L298N, can provide 2A continuous, each one can drive 2 motors.
        - For PIC programming either MPLAB X, or CCS C.
        - For voltage reduction for the mcu use: L7805CV
        - Use Proteus for code and electronics simulation.
    - Remarks
        - Battery choice will be made among Li-Po’s or Li-Ions, they will suffice.
        - pH Sensor or humidity sensors should be considered as actuators and we should only consider the data income not how the actuator will work.
        - GPS is not suitable for the application. Error margin is around a few meters, we need error of much smaller values. Future alternative: Farming grounds can be

outfitted with 3 small radio towers, a transmitter, receiver combo will let us triangulate the position much better.

        - A gyro sensor on board can also be added to map the terrain.
        - Possible future actuator addition: Some kind of thin rod to be inserted into the dirt every stop to “çapalamak” and aerate the roots increasing yield.
        - We should build a small prototype with constant ratio and make a proof of concept with it.
    - To do
        - Settle on an SD card module.
        - Have a look at encoders or other close control alternatives. (To be able to go straight on rough ground)
- PC software
    - Done
        - Take input: Rectangle length, height, width of each planting row, number of rows.

12.01.2021

- Structural
    - Done
        - SDG integration, done.
        - A semi-advanced RC car with suspension systems and somewhat stable cruising can be taken as a base for our structural model.
    - To do
        - Report draft.
- Electrical
    - Done
        - Proteus circuit of the robot.
        - Setup software for PIC programming with K150.
        - USE: MPLAB X IDE, XC8 compiler,
        - USE: A standard arduino SD card module. (There is nothing that ties it to the Arduino anyways, it can be used with MCUs.)
        - Order the components: PIC18F4431, 2 motor drivers TB6612FNG, SD Card module.
        -
    - To do
        - Integrate Embedded Code into Proteus simulation, make sure it works.

        - Build a few example circuits both in proteus and real life and code them in MPLABX to get used to PIC programming. (Simple LED circuit, Pwm drive, SD communication)
        - Have a look at the SPI protocol for the MCU and the SD Card.
        - Source the other components needed if any left.
        - Build Li-Po charging circuit for convenience.
- Embedded Software
    - Done
        - How algorithm flowcharts are made.
        - Abstract Code.
    - To do
        - Functional Code.
- PC Software
    - To do
        - Code flowchart.
        - Synchronization with Embedded software.

17.01.2021

- PC Software
    - Done
        - Conceptual algorithm.
    - To do
        - Orientation of the field should be added into the code.
        - Data output in the same way into
- Electronics+Embedded Software
    - To do
        - LED circuit.
        - Motor circuit.
        - SD circuit.

26.01.2021

- Redistribution of workload due to a group member's withdrawal of the course.
    - Modelling: Mehmet: Transfer function derivation.

13.02.2021

02.02.2021

- Electronics
    - Done.
        - PIC LED circuit and code.
        - PIC PWM circuit
    - To Do
        - PIC SD Card.
        - PIC Analog input.
    - Remark
        - Although right and left motors will be controlled in groups, differential drive can be achieved if wanted. (4 distinct outputs can be given).
        - Gyro sensor can be implemented to measure if there is uneven ground, allowing us to fix waterways for proper flow.
- PC Software
    - Done
        - Code flowchart.
        - Orientation of the field should be added into the code
        - Data output in the same way into

09.02.2021

- Electronics + Embedded
    - Debugged motor control code, now works as intended.
- Electronics
    - Formatted SDHC card into a FAT32 single partition, since it had been used elsewhere and needed this format.
    - Built the wooden chassis.

10.02.2021

- Debugged SD code, reformatted SD card, works as intended.
- Electronics
- Assembled robot with electronics and motors.

14.2 Appendix 2 - GitHub Pages

The Temperature Map Drawer in Java by Cem GEÇGEL:
https://github.com/Waistax/ME331HeatmapGenerator

The Embedded Code in C by Cem GEÇGEL:

https://github.com/Waistax/ME331EmbeddedSoftware

The Computer Software in Python by Çağrı ERGÜL:

https://github.com/cagriergull/Me331-Path-Finder

14.3 Demonstration Videos

- **https://youtu.be/QR4cUrX0dpw**
- **https://youtu.be/zXpWsx455rE**
- **https://youtu.be/dYCaRL1OT3Q**