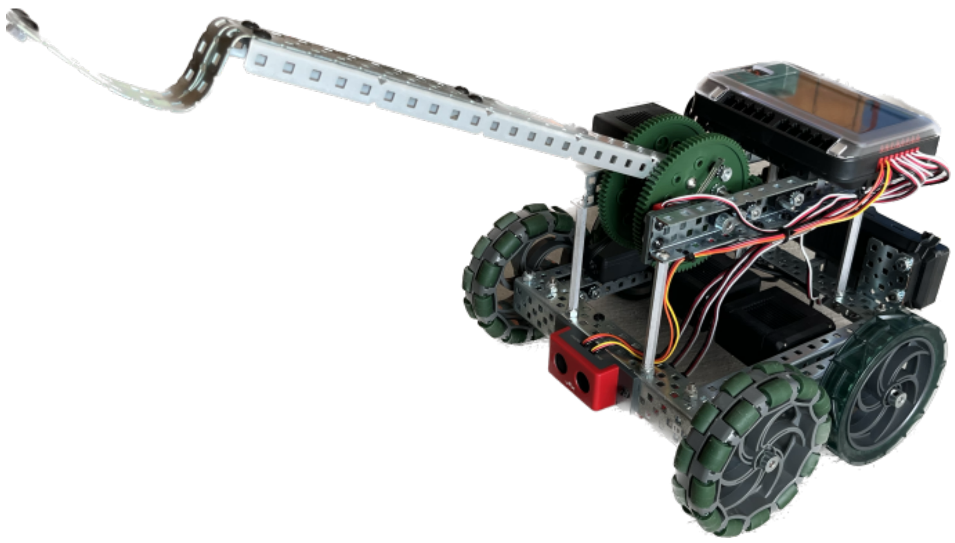
MECHENG 201

VEX Project – Part 1

Due in week 4 of the semester (worth 5%)

Name:	
Project Partner:	
Project Partner's Mobile Number:	
Lab Stream:	
Robot #:	



Semester 1, 2024

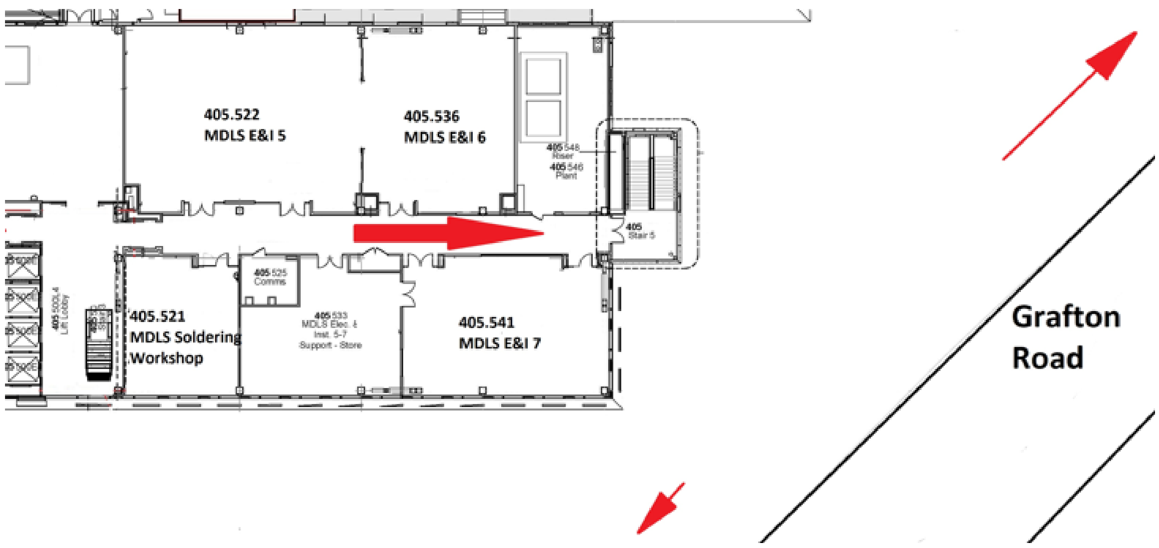
Safety Induction for B405-536 MDLS Electrical & Instrumentation 6

Welcome to MDLS E&I 6. This information has been provided to you because we are genuinely concerned about the health and safety of everyone here. As a University and an engineering student, we ask that you:

- Be properly attired – no Sandals, no jandals, no bare feet, or loosely fitting clothes.
- Secure long hair when approaching or working close to the machinery. Jewellery must be removed as with any loose items.
- Maintain proper decorum at all times.
- Be conscious of personal safety and of those working nearby at all times.
- Clean up equipment, tools, work area and surrounding floor after use.
- Stow away items, tables and chairs neatly at the end of the session.
- Report all injuries (including minor burns, bruises and cuts) immediately.
- Be familiar with the location of the first aid kits.
- Comply with the emergency Procedures – Note exit routes, location of phones, fire extinguishers.

Evacuation Plan

In the unlikely event of an emergency and immediate evacuation is initiated, please note the closest EXIT door of the learning space. The evacuation routes are shown (RED arrows) in the plan. The doors will be marked with a green coloured and lighted EXIT sign.



1 Learning Objectives

This project will introduce you to the basic principles of computer-controlled devices that can change a mechanical output based on sensor measurements from the environment. You will gain an appreciation for programming that deals with physical uncertainties. By completing Part 1 of the project, you will

- Learn how to interface with the VEX robot.
- Be able to write functions that control the basic movements of the VEX robot (driving, turning, and raising and lowering the arm) using open-loop control techniques.
- Find and understand the physical limitations of the the robot and open-loop control.
- Be able to draw complete flowcharts that describe the logic of your programming solution.

This part of the project is worth 5% of your final grade. You will need to demonstrate your solution during your second lab session in week 4. You will also need to submit your code and flowchart electronically to Canvas.

2 Background and Task Description

Due to the rising cost of labour, you are contracted by the Dairy Industry Cooperative to develop an autonomous robotic solution for delivering stock/payload (weighted cans). They want to test the autonomous solution first in a small storage room (Part 1 of the project) before testing the solution in their large warehouse (Part 2 of the project). A scaled down robot has been provided to you, along with a map (Figure 1) that represents features in the real world.

2.1 Objective

In the small storage room the automated forklift (VEX robot) is required to leave it's charging station indicated by *Start Area* (green hatched area) in Figure 1 to pick up the fragile payload (can) and to delicately place it somewhere in the drop-off zone before returning to the charging station (inside the yellow square in any orientation). The robot wheels must not leave the designated area bound by the warehouse walls (board outline) or collide with the stock shelves (red hatched areas), as doing so will risk damaging the property or equipment. The robot must start and end with the arm raised to the highest position.

2.1.1 Room Layout

A series of black lines are placed on the ground to be used as markers by the robot's light sensors. A brief description of the purpose of each line follows.

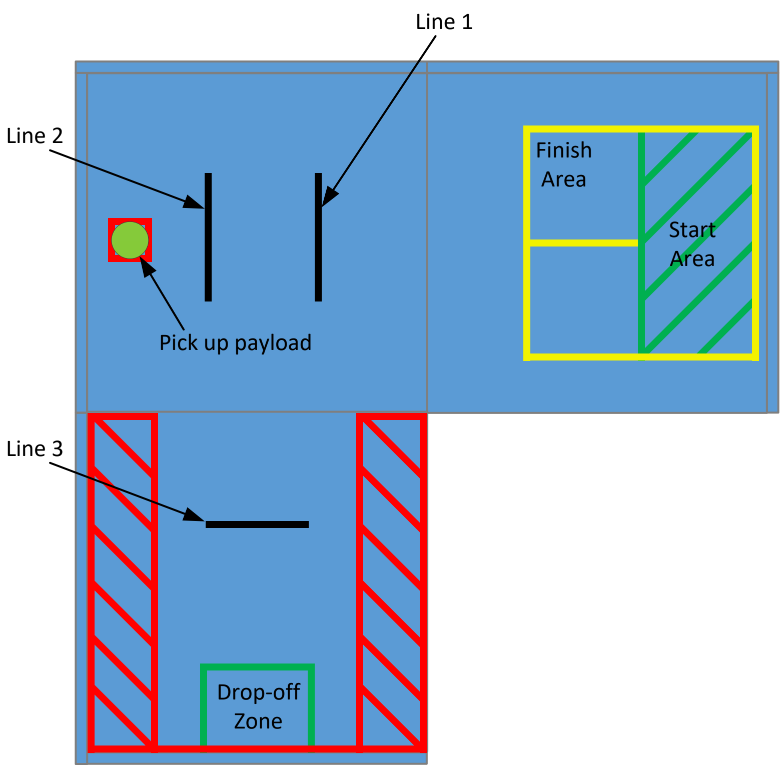


Figure 1: Small storage room layout.

- Line 1 indicates where the light sensors must be at such that robot arm can lift the payload *assuming* the arm has already been lowered to the correct height.
- Line 2 indicates where the robot must be at before initiating the left (counter-clockwise) turn to face the drop-off zone.
- Line 3 indicates the *approximate* position of the light sensors where the payload can be lowered into the drop-off zone.

3 Rules

1. All four wheels of the robot must start inside the start area (green hatched area) and finish inside the finish area (yellow box).
2. No further contact with the robot or can is permitted once the program has started.
3. The robot must start and end with the arm at its highest position.
   (a) For safety reasons, the arm must be raised shortly after dropping off the payload and before initiating any turning manoeuvrer.
4. You must use the light sensors to pick up and drop off the payload.
5. The robot must stop for 1 second after detecting each black line.

6. The arm and the wheels motors cannot be moving at the same time.
7. You are not allowed to use the encoders or the ultrasonic sensor for this part of the project.
8. The entire project code must be written in `Student_Code.c` and `Student_Code.h`.
9. You are only allowed to use the following commands when using the motors and sensors:
   - `motorPower(motorName,voltage);`
   - `armUp(voltage);` and `armDown(voltage);`
   - `readSensor(sensorName);`

4 Project Tasks

The project tasks are designed to work with each other to allow you to build the program to complete the final task. Therefore, we will emphasise the use of functions. Each task should be in its own function in addition to the supplementary functions you need to write. Everything should be contained in a single file (i.e. do not create a new file for each task). Use the template `VEX_Template_2024.zip` from Canvas as the starting point for your project.

4.1 Task 1 : Converting Motor Power [5 Marks]

Write a function called `convertPower()` that works according to the following specifications:

1. The only input is the percentage power level (`double` type) where 100% power corresponds to 5000 mV.
2. The function must use the `saturate()` function (see Appendix A) to limit the user inputs between -100 to 100 percent ***before*** converting the power.
3. It should return the converted voltage, in millivolts, as an integer number.

For example, an input of 20.5 to the function should return 1025 and an input of 205 should return 5000.

4.2 Task 2 : Detecting a Black Line [15 Marks]

Write a function `driveUntilBlack()` that takes one input (the percentage power) and drives forward at the specified percentage power and only stops when ***any*** of the three light sensors see the colour black. After detecting the black line and stopping the motors, the function should wait for one second before finishing. This function should not return any value. You must use the `convertPower()` function to convert the % power input to millivolts before sending the voltage to the wheel motors.

*Hint*s:

- Driving fast means that you can miss the line.
- Print the sensor readings to the LCD screen and place the light sensors on the black lines and record the range of sensor values corresponding to black lines.

4.3 Task 3 : Time-Based Arm Control [15 Marks]

Write a function called `armTime()` that takes two inputs according to the following specifications:

1. The first input is the percentage power level (`double` type) where the sign determines the direction of travel with positive values corresponding to raising the arm. You must use the `convertPower()` function to convert the % power input to millivolts before sending the voltage to the arm motor.
2. The second input is the time duration in milliseconds the arm motor has to be turned on (`int` type).

To test your `armTime()`function, you can write a program or function that raises the arm to its highest position, waits for 1 second, and then lowers it to an approximately horizontal position.

4.4 Task 4 : Time-Based Driving and Turning [15 Marks]

Write a function called `driveTime()` with three inputs according to the specifications below.

1. The first two inputs are the percentage power levels (`double` type) of the left and right wheel motors, respectively, where the sign determines the direction of travel with positive values corresponding to the forward direction. You must use the `convertPower()` function to convert the % power input to millivolts before sending the voltage to the wheel motors.
2. The third input is the duration in milliseconds the motors have to be turned on (`int` type).

This function can be used to drive and turn the robot for a specified duration at specified power levels for each wheel. If your robot does not drive in a straight line when given equal power to each motor, you can send slightly different powers to each motor to make it drive in an approximately straight line. Do NOT write a script to automatically correct for this power imbalance in this part of the project; this will be addressed in Part 2 of the project.

4.5 Task 5 : The Final Task [50 Marks]

Use the functions you have created in the previous tasks to write a program that will complete the main task of the project as described in Section 2 and according to the rules described in Section 3. To complete this task you need to do the following:

1. Complete the flowchart shown in Figure 2 (or start a new one) on a *new blank page*. Use estimates of powers and time durations to complete your initial flowchart; these will be updated once you write and test your program.

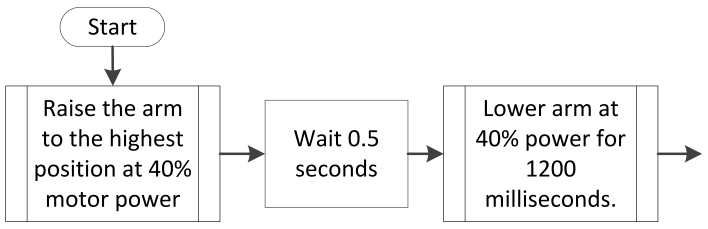

Start
Raise the arm to the highest position at 40% motor power
Wait 0.5 seconds
Lower arm at 40% power for 1200 milliseconds.

Figure 2: Example flowchart for the final task.

2. Write the program to execute the task according to your flowchart.
3. Test your program and update your values to complete the task.
4. Update the flowchart drawn in step 1 to represent the program that completes the task.
5. Demonstrate your solution to the lab TA.
6. Submit your code and flowchart to Canvas. See Section 5 for more details.

Hints:

- Do not use `armDown()` to lower the can!
- It is recommended that the robot waits for at least 1 second for the payload oscillations to reduce before lowering the payload. This gives greater accuracy when placing the payload on the ground.

5 Submission Requirements

Below are the submission and demonstration requirements for this project.

1. You need to demonstrate your complete solution (Task 5) to the lab TA during your second lab session in week 4 of the semester. They may ask you to show/test the other project tasks separately.
   - The marks for Tasks 1-4 will be based on the demonstration and code inspection during the lab session.
   - For Task 5, 25 marks are dedicated to the flowcharts (see next point).
2. After you demonstrate your solution to the lab TA, submit your code (`Student_Code.c` and `Student_Code.h` files only) and flowcharts (single PDF file that can have multiple pages) to Canvas by 11:59 pm on the day of your second lab session in week 4.

- Only one person from the group needs to submit on behalf of the group. The other person must check the submission for its correctness before the deadline.
- You do not need to upload the other files in your project.
- Your code must be well-commented and use good programming practices (e.g. meaningful variable names, not initialising variables inside a loop, etc.).
- The flowcharts must match your code (that's why you need to submit your code).
- You must draw a separate flowchart for each *subroutine you use* in your main flowchart for Task 5.
  - You do not need to draw the flowchart(s) for provided functions such as `armUp()` or `armDown()`.
- The flowcharts can be drawn neatly by hand, scanned, and uploaded to Canvas.
- The flowcharts should not contain code or function names. They should be in English and can contain mathematical expressions.

Late submissions will be penalised according to Table 1.

Table 1: Late submission penalties

How late?	Penalty
0 to 30 minutes late	5%
31 to 60 minutes late	10%
1 hour to 12 hours late	30%
12 hours to 1 day late	50%
More than 1 day late	100%

6 Useful Insights

This section is optional and will not contribute to your mark for this part of the project.

By addressing the issues and questions below, you will gain a deeper insight on the physical limitations of the VEX robot. This will come in handy when doing Part 2 of the project.

- Task 3: Repeat the task but with the robot carrying a payload. Do you get similar results? If not, how can we address that?
- Task 4: Do you need to provide each wheel a different power for your robot to travel in a straight line? If so, how can we deal with that automatically?
- Task 4: Try to drive back the same distance. Did the VEX robot return to its starting position? How does speed affect it? Does wheel slip affect the end result now that we're measuring wheel rotation?

- Task 2: Identify any potential issues associated with driving at low (10%) and high (90%) motor power levels with emphasis on the environment and mechanical design.

Appendix A The `saturate()` function

The `saturate(input, lower, upper);` function that takes three inputs and returns a single output according to the following specifications:

- The first input, `input`, is the number before saturation (`double` type).
- The second input, `lower`, is the lower saturation limit (`double` type).
- The third input, `upper`, is the upper saturation limit (`double` type).
- The function ensures that the output is saturated between these limits. If the input is between these limits, the output is equal to the original input. If the input is outside the saturation limits, the the output is limited at the corresponding saturation limit as shown in Figure 3.

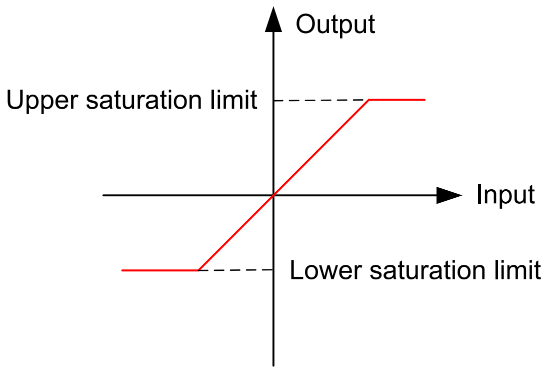


Figure 3: Output saturation