Dexter : The Capability Chair

Design

March 27, 2017

Mechatron 4TB6 - Dr. Alan Wassyng

Group 4:

Jose Bonicelli	1211093
Vaishnovi Patel	1208768
Neil Hsieh	1216670
Joshua Havin	1208642
Ethan D'Mello	1217558
Andrew Chin	1203190
Gavin Johnson	1217930

Revision History

Version.	Date	Description	Author
Rev. 0	Nov.14 2016	Finalized version for System Requirements deliverable.	ALL
Rev 0.1	Dec.15 2016	Edited and appended to Constants section. (max_..._pwm)	Neil
Rev 0.2	Dec. 15 2016	Addition of section for individual Component I/O behaviour.	Neil & Josh
Rev 0.3	Dec. 15 2016	Added Components section.	Wish
Rev 0.4	Dec. 16 2016	Added Normal Operation and Undesired Event Handling sections	Andrew
Rev 0.5	Dec. 17 2016	Changed the Context Diagram	Wish
Rev 0.6	Dec. 17 2016	Appended Component Behaviours to Component sections.	Neil & Josh
Rev 1	Mar. 23 2017	Finalized version for System Design deliverable	ALL

Table of Contents

List of Figures

List of Tables

1. Purpose

1.1 Document Purpose:

This document is meant to outline the design behind Dexter: a fully functional capability chair. The content provided within will explain how Dexter interacts with its environment, and what is defined as normal operation for the system. The document is broken down into two parts; sections 3 to 10 will comprise of an overall system design and sections 11 and 12 will examine the components in further detail. Throughout the document, issues and constraints will be analyzed in order to support all design decisions.

1.2 Design Purpose

Today, all public Canadian buildings are required by law to be accessible by people with disabilities. [1] This wonderful reality promotes equality and fair practice for persons with reduced mobility in the modern world. That being said, accessibility does not equate to convenience, and extra time and effort can become barriers in an inconvenient, yet accessible work environment. This unfortunate fact has led us to search for a solution to solve this unique problem: “Make All Workplaces Convenient.”

Perhaps an example of environment-based hardship will sculpt a little more detail into this ambiguous problem. In Automobile Workshops, excellent mobility is essential. Workers require dexterity to operate heavy machinery when accomplishing precision-oriented tasks. Mechanics find themselves in constant motion in order to diagnose, repair, disassemble and assemble various components of vehicles. Traditional wheelchairs and powered mobility scooters are often found at a disadvantage in this scenario for two, very different reasons:

1. Mobility scooters only require one hand to operate, and move quickly, but they must be driven like a car. On-axis rotation is not possible and the user must turn while using a combination of forward and reverse, which can pose a problem in the narrow confines of a workshop.

2. Wheelchairs are able to change orientation without changing position, but this requires two hands. Workers would not be able to turn or move without dropping what they are holding.

Both of these points indicate that there must be a better way.

2. Scope

2.1 Project Scope

It is impossible (not to mention inefficient) to change and enlarge every thinkable workplace to accommodate persons with reduced mobility, therefore a solution is much better founded in improving the ability of the aforementioned individuals instead of adapting the environment. Our scope lies within designing a self-contained mode of transportation that will easily navigate through cluttered environments. In order for it to fulfill this purpose it must satisfy the following criteria:

1. Rotate on axis
2. Move forwards & backwards
3. Move laterally
4. Fit through a standard doorway
5. Carry the user
6. Be fully controlled by the user
7. Be completely self-contained

The environment is assumed to be wheelchair accessible, therefore stairs and rough terrain handling would be considered out of scope. That being said although sharp inclines are not explicitly involved they are important to consider, as ramps and uneven terrain are very common in many work environments. Since this is a mode of transportation designed for the workplace, the design will have to include considerations to prioritize the safety of the operator.

2.2 Project Goals

Goal	Measurable/Quantifiable Aspect or Method
Keep the project safe	All aspects of the project will be reviewed by the McMaster board of ethics
Keep all code tidy	Comments & Code diagrams will be drawn
Make Dexter aesthetically pleasing	Polished design with minimal exposed components
Make Dexter durable	Keep the device components protected, use materials traditionally deemed as strong (i.e. reduce use of plastic)
Make Dexter environmentally friendly	Use as many recycled materials as possible,

	refrain from using plastics
Make Dexter comfortable	Several test users will evaluate the device on comfort based on ergonomics and smoothness of ride
Spend as little money as possible	The budget is a maximum, not a goal
Keep users from getting dizzy	Limit turning speed and emphasize the smoothness on all rotational movements
Make motion smooth in every direction	Limit acceleration to an appropriate amount, determined via trial and error

Table 1: Project Goals

2.3 Road Map and Background

This document was initially prepared before the building process, but some revisions have been made as dexter has begun to take shape. For instance, upon building the prototype chassis it was noticed that there was an excessive amount of torque experienced by the bearing when the axle was only supported at one end. This lead to a re-design to balance the rotational forces as explained in section 11.1.1. At this revision the build is nearly complete, and all design decisions can be considered virtually final.

3. Context Diagram

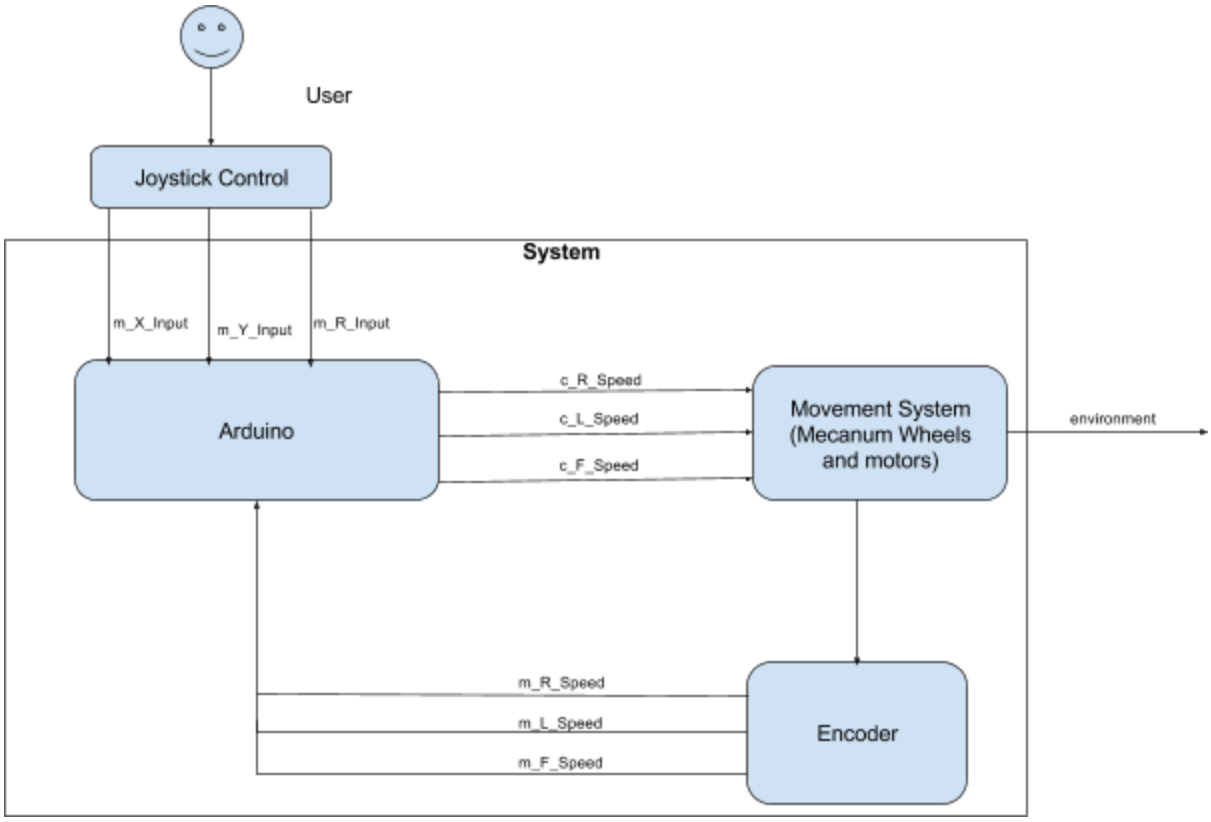


Figure 1: System diagram with inputs and outputs

4. Monitored and Controlled Variables

Name	Description	Range	Unit	Type
m_R_Speed	Observed rotational speed	N/A	RPM	Monitored
m_L_Speed	Observed lateral speed	N/A	m/s	Monitored
m_F_Speed	Observed forward speed	N/A	m/s	Monitored
c_R_Speed	Target rotational speed	-10 to 10	RPM	Controlled
c_L_Speed	Target lateral speed	-2 to 2	m/s	Controlled
c_F_Speed	Target forward speed	-2 to 2	m/s	Controlled

m_R_Input	The rotational displacement	-10 to 10	rpm	Monitored
m_Y_Input	The forward/backward displacement indicated by the user	-2 to 2	m/s	Monitored
m_X_Input	The lateral displacement indicated by the user	-2 to 2	m/s	Monitored
BatteryCharge	Is the battery providing power to the device	0,1	N/A	Monitored

Table 2: Monitored and Controlled Variables

5. Constants

Name	Description	Value	Units
k_Max_Velocity	Maximum achievable linear velocity for motor output.	± 2	m/s
k_Max_Supported_Weight [3]	Maximum supported weight for 4 combined wheels.	90	kg
k_Max_Angular_Velocity	Maximum achievable angular velocity for system.	10	rpm
max_lvelocity_pwm	Arduino analogWrite when maximum linear velocity is achieved. Value in terms of value written to analogWrite.	TBD	n/a
max_avelocity_pwm	Arduino analogWrite when maximum angular velocity is achieved. Value in terms of value written to analogWrite.	TBD	n/a

Table 3: Behaviour Constants

6. Components and Behaviours:

This section will describe the individual components that collectively make up the system as a whole. There are 3 components: Central Controller, Encoder, and the Movement System. Included are role overviews, behaviours of each component, respective inputs/outputs, and initializations.

6.1 Central Controller

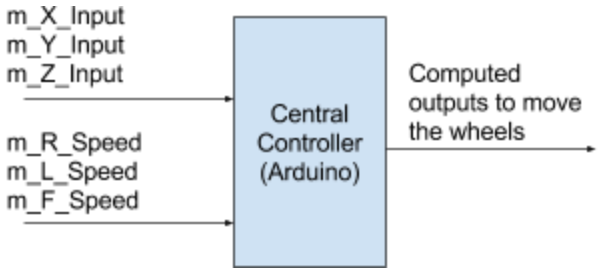


Figure 2: Central Controller Block Diagram

This component is responsible for receiving and processing all the input responses from the joystick and the feedback response of the motor outputs. It also is responsible for translating the user input into commands for the movement system. It handles all of the processing and decision making in the system.

Central Controller Behaviour	
Inputs	m_X_Input, m_Y_Input, m_Z_Input, m_R_Speed, m_L_Speed, m_F_Speed
Outputs	Computed outputs to each individual motor to power wheels
I/O Relationship	The outputted instructions to the motors are dependent on a function of the inputs and the current motor speeds.
Timing Constraints	Operate at a speed such that no interrupts on any of the inputs are missed.

Initialization	m_X_Input = 0;
	m_Y_Input = 0;
	m_Z_Input = 0;
	m_R_Speed = 0;
	m_L_Speed = 0;
	m_F_Speed = 0;

Table 4: Central Controller Behaviour Breakdown

6.2 Joystick Input

Description:
Physical user interaction with the joystick input is translated to digital information labeled as “User Input”. User input will be processed by the central controller which will convert the directional input to voltages which are then delivered to the wheel motors based on the perceived user input(s). Voltage outputs will be based on three (3) different fundamental wheel configurations: linear movement (forwards and backwards), lateral movements (left and right). angular movements (rotation about an axis).

Joystick Behaviour	
Inputs	User input on 2 Joysticks (One with x,y output and the other with simply z output, meant to control rotation)
Outputs	m_X_Input, m_Y_Input, m_Z_Input
I/O Relationship	Each value is proportional to the amount that the user moves the joystick in its respective direction (x, y or z)
Timing Constraints	Operate at a speed such that no interrupts on any of the joystick inputs are missed.
Initialization	m_X_Input=0, m_Y_Input=0, m_Z_Input=0

Table 5: Joystick Behaviour Breakdown

6.3 Encoder

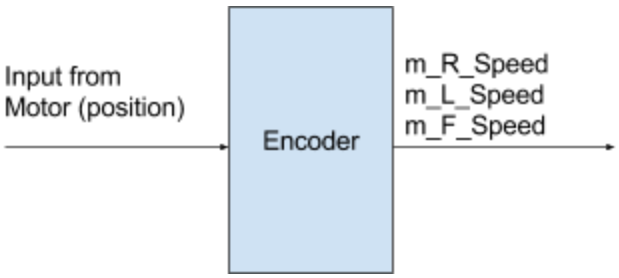


Figure 3: Encoder Block Diagram

Description:
Encoder output will be processed by the component for use in speed monitoring, in particular to check if system has reached maximum linear and/or angular velocity. Obtains motor position/speed as inputs and outputs the measured variables to the central controller.

Motor Feedback Behaviour	
Inputs	Wheel movement (From motor)
Outputs	To Arduino Controller: m_R_Speed, m_L_Speed, and m_F_Speed
I/O Relationship	The encoder reads each motor's individual velocities which are directly proportional to the rotational velocity of each wheel & converts it into values processable by the microcontroller
Timing Constraints	Operate at a speed such that no interrupts on any of the outputs are missed.

Initialization	m_R_Speed = 0;
	m_L_Speed = 0;
	m_F_Speed = 0;

Table 6: Motor Feedback Behaviour Breakdown

6.4 Movement System

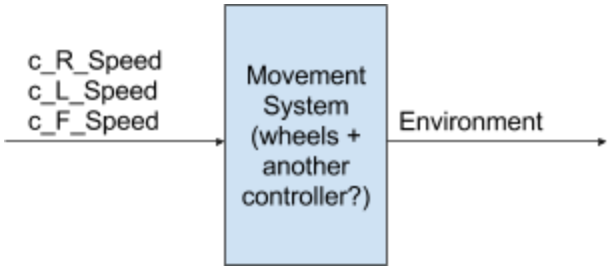


Figure 4: Movement System Block Diagram

Description:
This component takes the inputs from central controller in the pre-configured wheel configurations and sends the respective voltages to each corresponding motor.

Movement System Behaviour	
Inputs	c_R_Speed, c_L_Speed, c_F_Speed
Outputs	Voltage to respective motor
I/O Relationship	The output voltage and movement is directly proportional to the requested speed
Timing Constraints	Operate at a speed such that no interrupts on any of the inputs/outputs are missed.

Initialization	c_R_Speed = 0;
	c_L_Speed = 0;
	c_F_Speed = 0;

Table 7: Movement System Behaviour Breakdown

7. Performance Requirements

7.1 Translational Velocity

Description	The device shall move at a speed comparable to existing mobility devices.
Rationale	The user should not be able to travel past a maximum speed in order to remain safe and in control of the device.
Importance	5/5 - Safety Critical
Unit	meter per second (m/s)
Ideal Value	2 m/s

7.2 Rotational Velocity

Description	The device shall rotate at a speed comparable to existing products.
Rationale	The user should be able to rotate past a maximum angular speed in order to remain safe, in control of the device, and avoid disorientation of the user.
Importance	5/5 - Safety Critical
Unit	Degrees per second (deg/s)

Ideal Value	60 deg/s

7.3 Translational Acceleration

Description	The device shall not accelerate at a greater value than half the current velocity of the device.
Rationale	The acceleration of the device will be limited relative to the current velocity in order to minimize the jerk while changing velocity.
Importance	4/5 - Safety Critical
Unit	meter per second per second (m/s^2)
Ideal Value	<1 m/s^2

7.4 Angular Acceleration

Description	The device shall not have angular acceleration at a greater value than half the current angular velocity of the device.
Rationale	The acceleration of the device will be limited relative to the current velocity in order to minimize the jerk while changing orientation and to help the user keep balance.
Importance	4/5 - Safety Critical
Unit	degrees per second per second (deg/s^2)
Ideal Value	<30 deg/s^2

7.5 Translational Resolution

Description	The device shall move a minimal distance in order to make minute adjustments.
Rationale	The user should be able to only move the desired distance so a small resolution is required to avoid overshoot.
Importance	3/5 - Not Safety Critical
Unit	meter
Ideal Value	The smallest movement possible is less than 0.05m

7.6 Angular Resolution

Description	The device shall move a minimal angle in order to make minute adjustments.
Rationale	The user should be able to only rotated to the desired orientation so a small resolution is required to avoid overshoot.
Importance	3/5 - Not Safety Critical
Unit	degree(deg)
Ideal Value	6 deg

7.7 Battery Charge

Description	The device shall operate for a minimum certain distance on a single full charge of the battery.
Rationale	The user should be able to operate the device for a distance sufficient to get work accomplished before needing to recharge the device.
Importance	4/5 - Functionally Critical
Unit	meter(m)
Ideal Value	1000m

8. Behaviour States

8.1 Idle (stationary)

The device is powered and the motors keep the device stationary. This occurs when m_R_Input, m_L_Input, m_F_Input, m_R_Speed, m_L_Speed, m_F_Speed are all equal to zero and the device is receiving power; the power applied to the motor changes so that m_R_Speed, m_L_Speed, and m_F_Speed are equal zero. When any of m_R_Input, m_L_Input, or m_F_Input change to have a positive magnitude, the state will transition to become Displacement. When the device stops receiving power, the state will transition to become Off.

Initialization:
State = 0;
BatteryCharge = 1;
m_R_Speed = 0;
m_L_Speed = 0;
m_F_Speed = 0;
c_R_Speed = 0;
c_L_Speed = 0;
c_F_Speed = 0;

Timing Constraints: Operate at a speed such that no interrupts on any of the inputs/outputs are missed.

8.2 Displacement

The device is powered on and the motors are moving the device. The speed and direction are determined by the magnitude of c_R_Speed, c_L_Speed, c_F_Speed; the power applied to the motor changes so that m_R_Speed, m_L_Speed, and m_F_Speed are equal to c_R_Speed, c_L_Speed, and c_F_Speed, respectively. When any of m_R_Input, m_L_Input, or m_F_Input change to be zero, the state will transition to become Coast. When the device stops receiving power, the state will transition to become Off.

Initialization:
State = 1;
BatteryCharge = 1;
m_R_Speed = 0;
m_L_Speed = 0;
m_F_Speed = 0;
c_R_Speed != 0;

c_L_Speed != 0;
c_F_Speed != 0;

Timing Constraints: Operate at a speed such that no interrupts on any of the inputs/outputs are missed.

8.3 Coast

The device is powered on and the motors are spinning freely. This occurs when m_R_Input, m_L_Input, m_F_Input, are all equal to zero but any of m_R_Speed, m_L_Speed, or m_F_Speed are not zero and the device is receiving power. When any of c_R_Speed, c_L_Speed, or c_F_Speed change to have a positive magnitude, the state will become Displacement. When any of m_R_Speed, m_L_Speed, or m_F_Speed change to be zero, the state will transition to become Idle. When the device stops receiving power, the state will transition to become Off.

Initialization:
BatteryCharge = 1;
State = 2;
m_R_Speed != 0;
m_L_Speed != 0;
m_F_Speed != 0;
c_R_Speed = 0;
c_L_Speed = 0;
c_F_Speed = 0;

Timing Constraints: Operate at a speed such that no interrupts on any of the inputs/outputs are missed.

8.4 Off

The device is powered off. This only happens when the is no power being supplied to the device, whether the battery has no charge or the power switch if off. When the device starts receiving power again, it will go into the Idle state.

Initialization:
State = 3;
BatteryCharge = 0;
m_R_Speed = 0;
m_L_Speed = 0;
m_F_Speed = 0;
c_R_Speed = 0;
c_L_Speed = 0;

c_F_Speed = 0;
Timing Constraints: No timing constraints, only required variable is a hardware interrupt.

9. Normal Operation

The device is intended to be used as a mobility aid in workplaces and other environments where space is limited, and conventional wheelchairs would have difficulty turning. In the course of use, the device is expected to perform on flat and inclined surfaces of relatively smooth terrain. Under normal use conditions, the device will allow users to move forwards, backwards, and strafe in any direction. The user will also be able to rotate clockwise and anti-clockwise while maintaining forward motion.

Normal Use Cases:

1. System is powered off

When the system is powered off, the device will remain in a stationary position and will not respond to user input.

2. System is powered on and stationary without user input

When the system is powered on and stationary without user input, the device is on standby and is able to receive and react to user input or power-off

3. System is powered on and stationary with user input

When the system is powered on and stationary and receiving user input, the device will begin to move in the way specified by the user input. Eg. if the user inputs a forward input from the controls, the device will begin to move in the forward direction, at a controlled rate of acceleration

4. System is powered on and mobile with user input

When the system is powered on and mobile with user input, the device will alter its current course and speed at controlled rated based on the new user input being received

5. System is powered on and mobile without user input

When the system is powered on and mobile without user input, the device will gradually reduce its current velocity to zero and will become stationary, awaiting further input or a power-off

10. Undesired Event Handling

Undesired events in the system can arise from 2 sources; unexpected user input and device failure. The system can constrain user input to avoid some undesired events. This will be done by limiting speed and controlling acceleration for a smooth and safe ride.

Undesired Event	**Event Handler**	**Monitored Variables**	**Controlled Variables**
User attempts to accelerate beyond maximum speed	Speed of device is limited and acceleration input is ignored	m_X_Input; m_Y_Input; m_R_Input; m_F_Speed; m_R_Speed; m_L_Speed	c_F_Speed; c_R_Speed; c_L_Speed
Device is descending an incline	Considered out of scope, may be approached in future revisions.	N/A	N/A
Device experiences power failure while in motion	Device shuts down and coasts to a stop; wheels will not lock.	N/A	N/A
Essential component failure	Device shuts down; wheels will not lock.	N/A	N/A

Table 8: Undesired Event Handling

11. Module Guide

11.1 Mechanical Components

11.1.1 Chassis

Functions: There are two requirements that are completely dependent on the chassis for the device to function properly; the chassis must support the user and it must contain all of dexter's components. The third function of the chassis is implied by the other two, in the sense that it must accomplish these tasks all while protecting the user from its moving components, and the components themselves from the user and the external environment.

Materials: A combination of an aluminum alloy skeleton and exoskeleton and a hobby-grade wood board were chosen to ensure the device's flexibility and strength. The device will be assembled using a combination of screws, nuts and bolts.

Structure design: The skeleton and exoskeleton are shown in Figure 5. The purpose of the skeleton is to provide stability and rigidity for the mecanum wheel and motor system. It also provides mounting and support for the wheels, motor and gear drivetrain. The purpose of the exoskeleton (which encompasses the metal bars surrounding the mecanum wheels) is to provide additional strength to the chassis so that Dexter can support the maximum stated weight of 200 lbs. Additionally, the exoskeleton provides "bump and break" protection for the wheels, so that Dexter's integrity remains intact in the event of a crash.
Where the purpose of the internal chassis is to provide strength and an inner structure to mount the hardware, the hardwood board will be used as added reinforcement, and to support the user. For this reason, it is important to go for hobby-grade wood, as it is less likely to warp, ensuring a proper fit with the coinciding metal pieces. The hardwood board will cover the entirety of the skeleton, up to the mecanum wheels, leaving the exoskeleton bare. The points of contact between the board and the skeleton will be bolted and will encompass the perimeter and four central points to minimize wood warping.

Mathematical Justification: The exoskeleton provides load dissipation by adding another support point to the wheel axle. This effectively removes the torque experienced at the inside bearing by balancing the payload to another bearing at the opposite end of the axle, while halving the total force at each bearing. The free body diagrams demonstrating this is found in Figure 6.

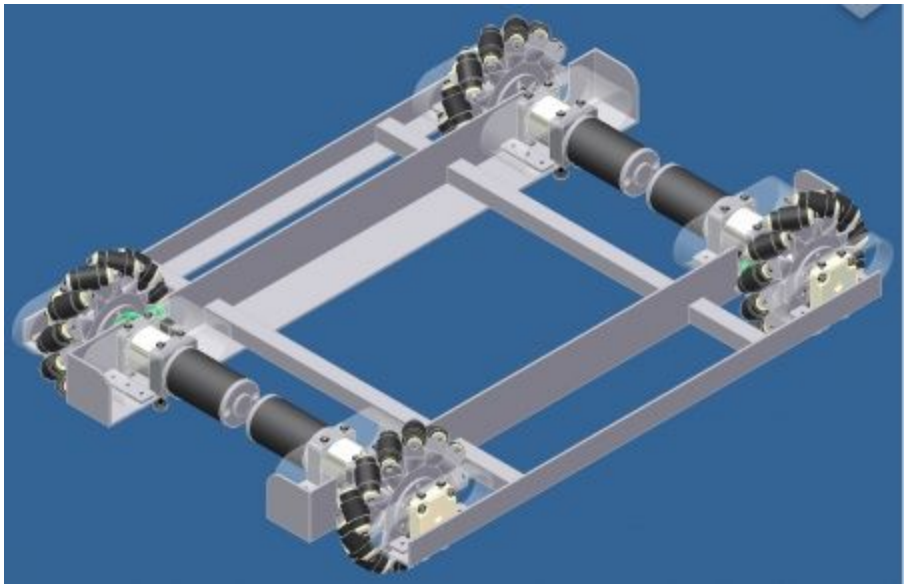


Figure 5: Chassis model

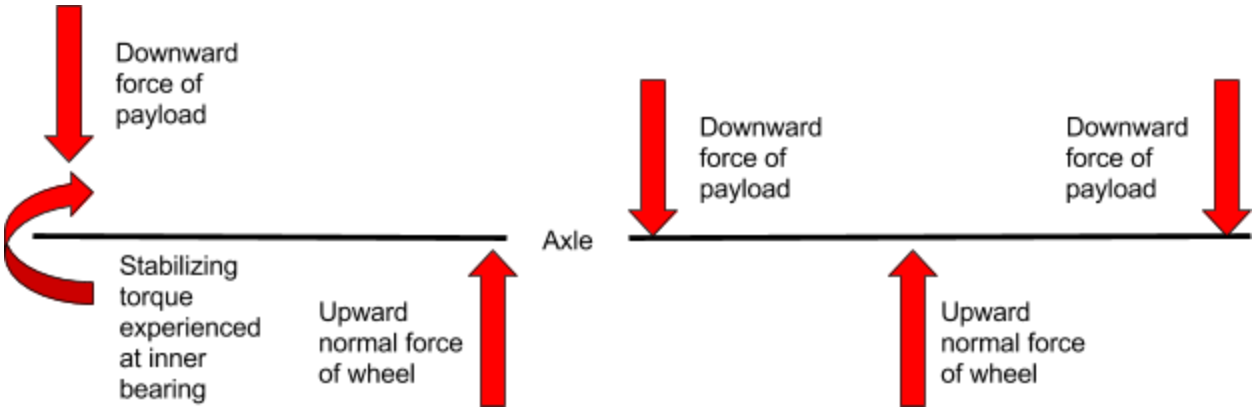


Figure 6: Free body diagrams comparing two axles. Left is undistributed without exoskeleton and right is with dexter's exoskeleton implemented.

11.1.2 Seating System

Functions: In a sense, the seating system is just a glorified chair, but its functions span further than just providing the user with a comfortable seat. According to our hazard analysis, the chair must secure the user and protect them from falling due to sudden movements. Additionally, it must give the user ergonomical access to the Joystick controls.

Materials: For revision 0, the seating system will be very bare bones in order to accommodate any required improvements/upgrades. In its initial state, the chair will simply be a standard folding chair bolted to the wood board of the chassis, with a seat belt system for safety

.

Structure: For revision 0, a pre-built standard folding chair will be used. This was chosen due to it's lightweight, and modular design, making it perfect for the preliminary testing phase. Since folding arm chairs do not have armrests, the joysticks will not be mounted to the chair, and instead will be free floating, connected only to the dexter by the appropriate information cables. Special consideration will have to be taken into place in order to protect these cables and their connections, but this should not affect the overall functionality of the device, as the user will have to hold the controller in the same way that they would hold a wired gaming controller.

11.1.3 Gear Drivetrain

Function: The gear drivetrain is used to effectively reduce amount of torque required to push around the payload, as well as to isolate the motor from the downward force of the payload.

Materials: Aluminum gears will be used for low weight durability.

Structure: The Mecanum wheels will be installed onto the chassis using a gear train that will have a one – five gear ratio. The gear train will consist of 2 gears with the first gear connected to the motor and the second gear connect to the wheel via a shaft and hub. This will prevent too much force being exerted onto the motor axle which could break the motor. We explored using a chain instead of gears however we wanted flexibility in changing the gear ratio to allow for different speeds and torques.

Mathematical Justification: The gear ratio is obtained by comparing the 12t motor gear to the 60t wheel gear. 12:60 can be simplified to 1:5.

11.1.4 Mecanum Wheels

Function: The wheels that we use for our device are key to the functionality of our device, allowing dexter to move in virtually any way imaginable. The movement possibilities and corresponding wheel directions are illustrated in Figure 9.

Materials: Our team is using 6-inch Mecanum wheels that we purchased from Vex-Robotics. The wheels we are using are conservatively used to hold a weight of 200 lb.

Mathematical Justification: A Mecanum wheel is based on the principle of a central wheel with a number of rollers placed at an angle around the periphery of the wheel. The angle between rollers axis and central wheel axis could have any value, but in the case of conventional Mecanum wheel it is 45°. The rollers are shaped such that the silhouette of the omnidirectional wheel is circular. The angled peripheral rollers translate a portion of the force in the rotational direction of the wheel to a force normal to the wheel direction. Depending on each individual wheel direction and speed, the resulting combination of all these forces produces a total force vector in any desired direction, thus allowing the platform to move freely in direction of the resulting force vector, without changing the direction of the wheel.

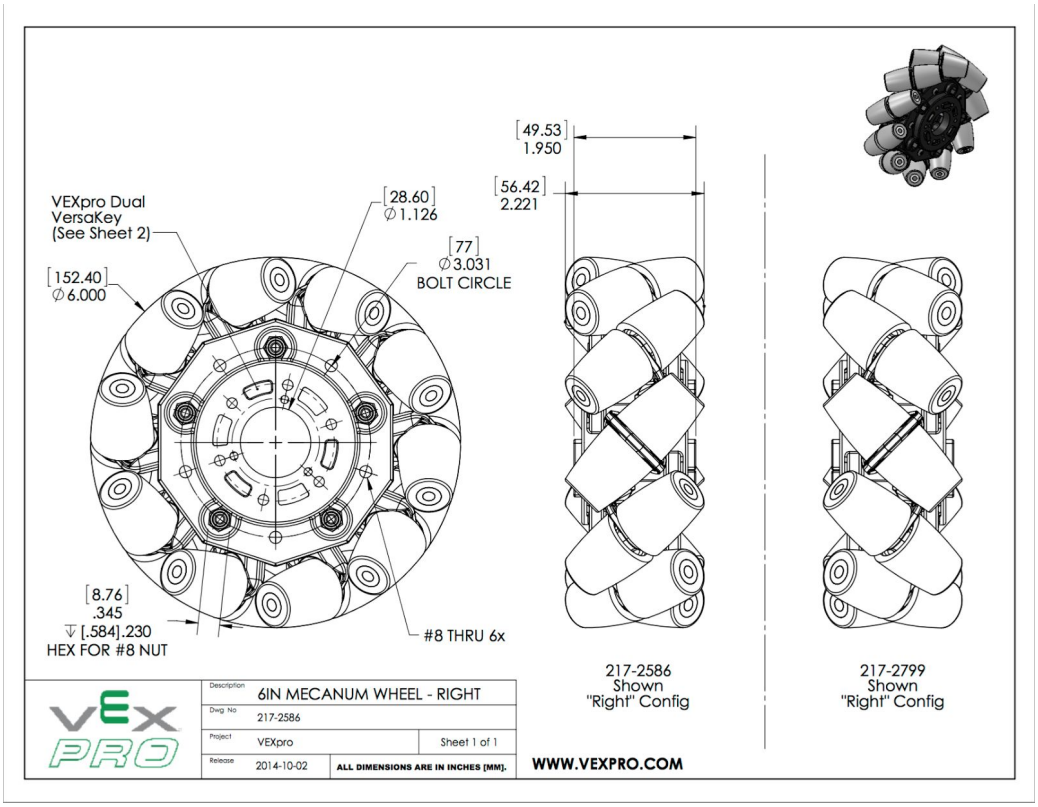


Figure 7: CAD Drawing of Mecanum Wheel [3]

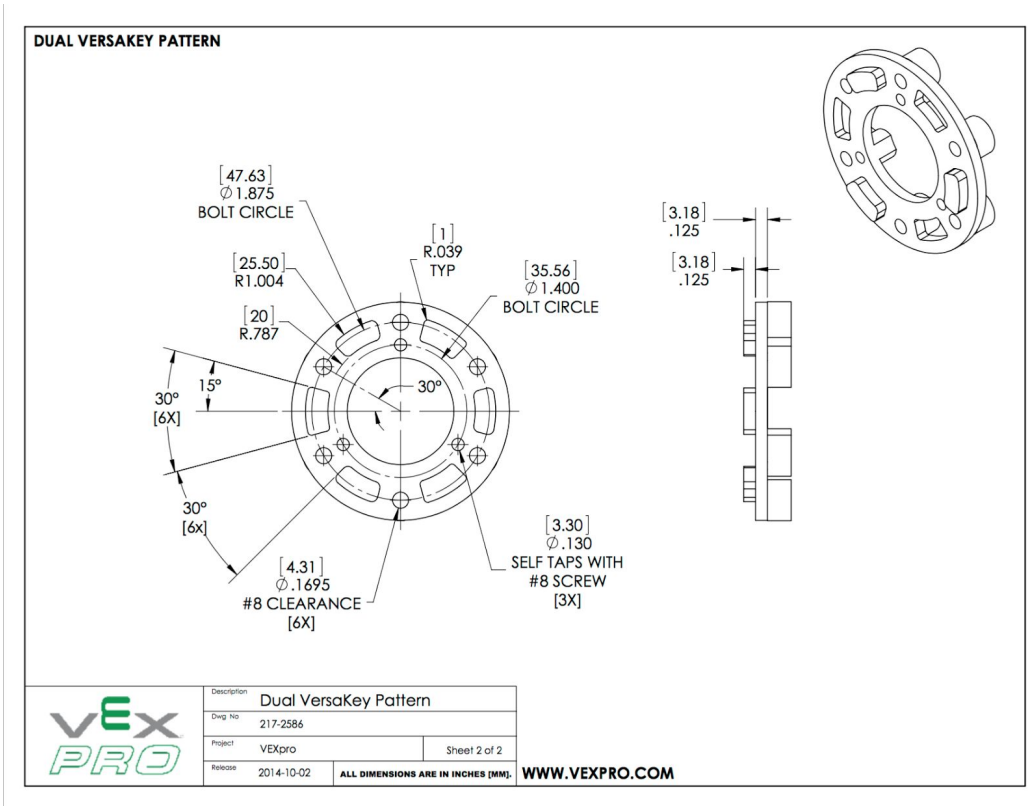


Figure 8: Technical Diagrams of Mecanum Wheel Hub [4]

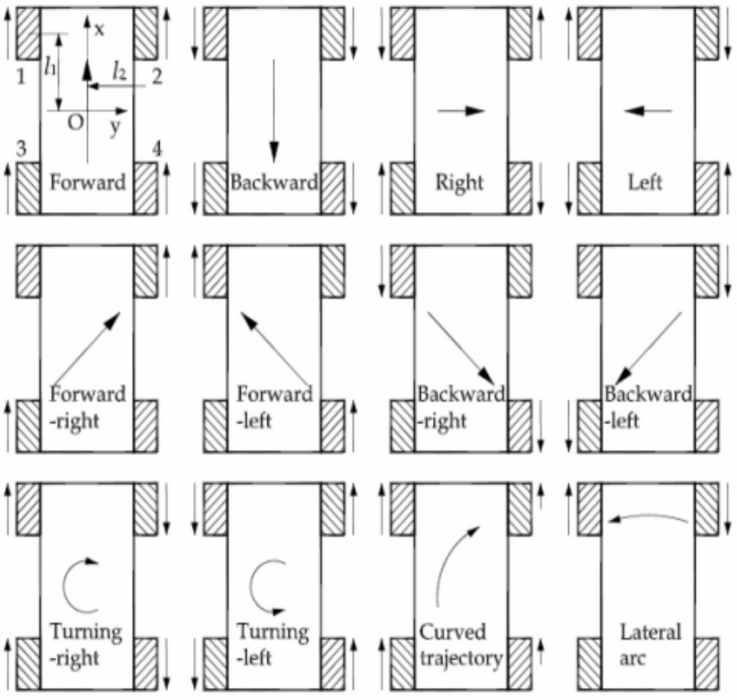


Figure 9: Wheel Direction and Related Movement [5]

11.2 Hardware Components

11.2.1 Microcontroller - Arduino Mega

The brain of the system, which will be used to gather information from the environment, the user and respond accordingly. Arduino Mega has an operating voltage of 5V but it can be powered using up to 7-12V USB connection or an external power supply. One of the biggest reasons why we chose to go with the Mega is the number of analog and digital pin inputs offered for performing tasks. It is low powered and sufficient for performing all the functionalities dedicated by the Dexter.

Input:

- 12V Lithium Ion Battery
- From the Encoders
- Sensors

Output:

- Signals to the Speed Controller
- Sensors

11.2.2 Motor Drive System

The Motor Drive system will work in harmony with a set of 4 DC motors all connected to their corresponding speed controllers which would moderate their speed based on a PWM signal sent from the Arduino. Attached to the motor is an encoder which is responsible for sending motor position feedback to the Arduino and through simple calculation, the speed of each mecanum wheel, ergo the speed of the system would be known.

The Arduino will take the encoder information along with the user input from the joystick and send appropriate signals back to the speed controller to either speed up or slow down the corresponding wheel(s).

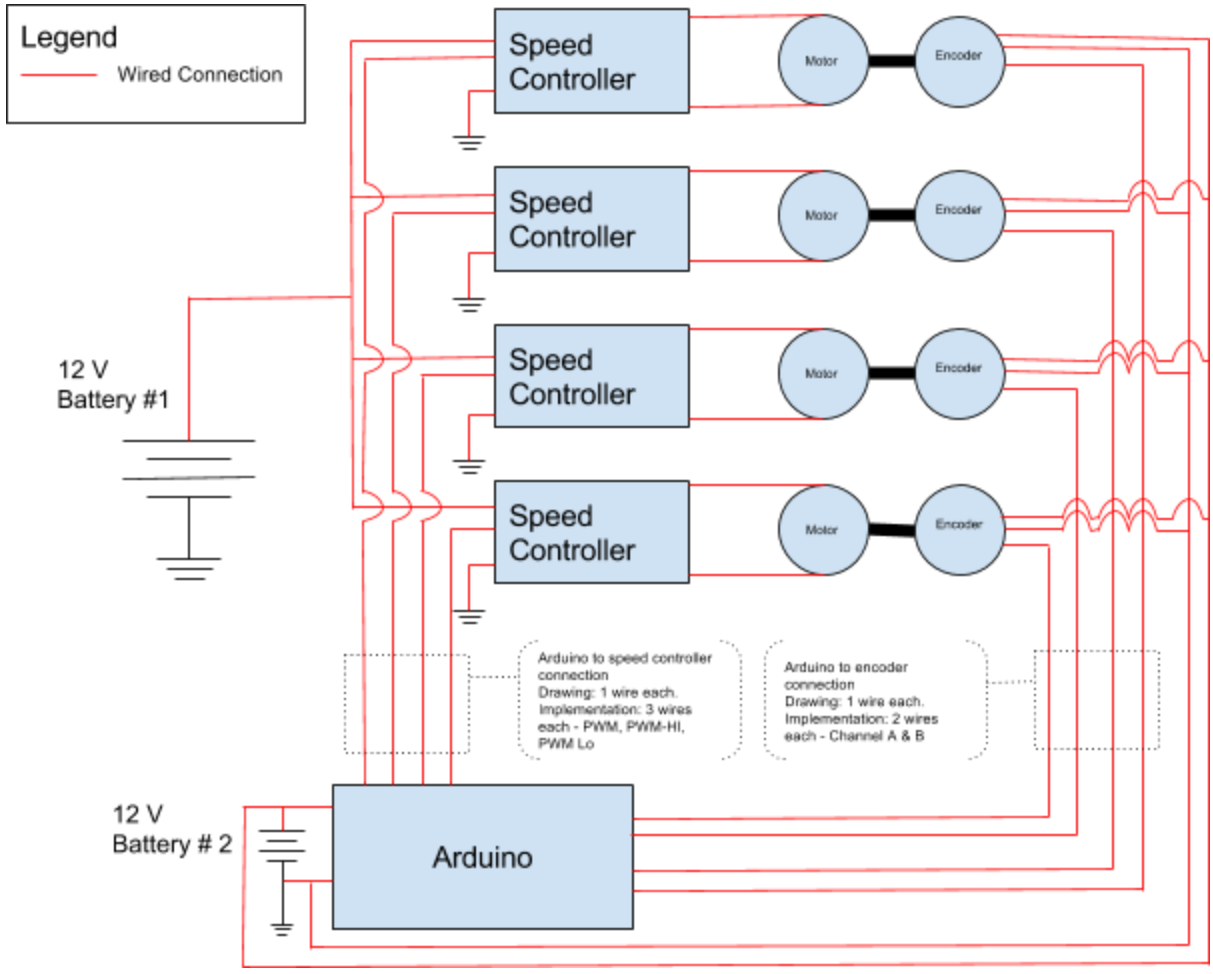


Figure 10: Motor Drive Circuit

Mathematical Justification:

To calculate the compatibility of the torque required by the system, we had to also include torque calculations for the motor, and see the torque output possible. After the gear ratio, our motors were found to be able to move our system with additional torque that may be required. Equations could be seen below.

$$\tau_{motor} = \frac{Power\ x\ Efficiency}{RPM}\ x\frac{60}{2\pi} = \frac{\mathbf{300\ x\ 0.70}}{\mathbf{250}}\ \boldsymbol{x}\frac{\mathbf{60}}{\mathbf{2\pi}} = \mathbf{8.02\ Nm}$$

$$With\ gear\ ratio\ 5{:}1 \rightarrow 8.02\ x\ 5 = \mathbf{40.12\ Nm}$$

$$\tau_{system} = Mass_{system}\ x\ Acceleration_{system}\ x\ Radius_{wheel}$$

$$\tau_{system} = \mathbf{90}\ kg\ x\ \mathbf{1}\ {}^{m}/_{s^2}\ x\ \mathbf{0.0762}\ m = \mathbf{6.858\ Nm}$$

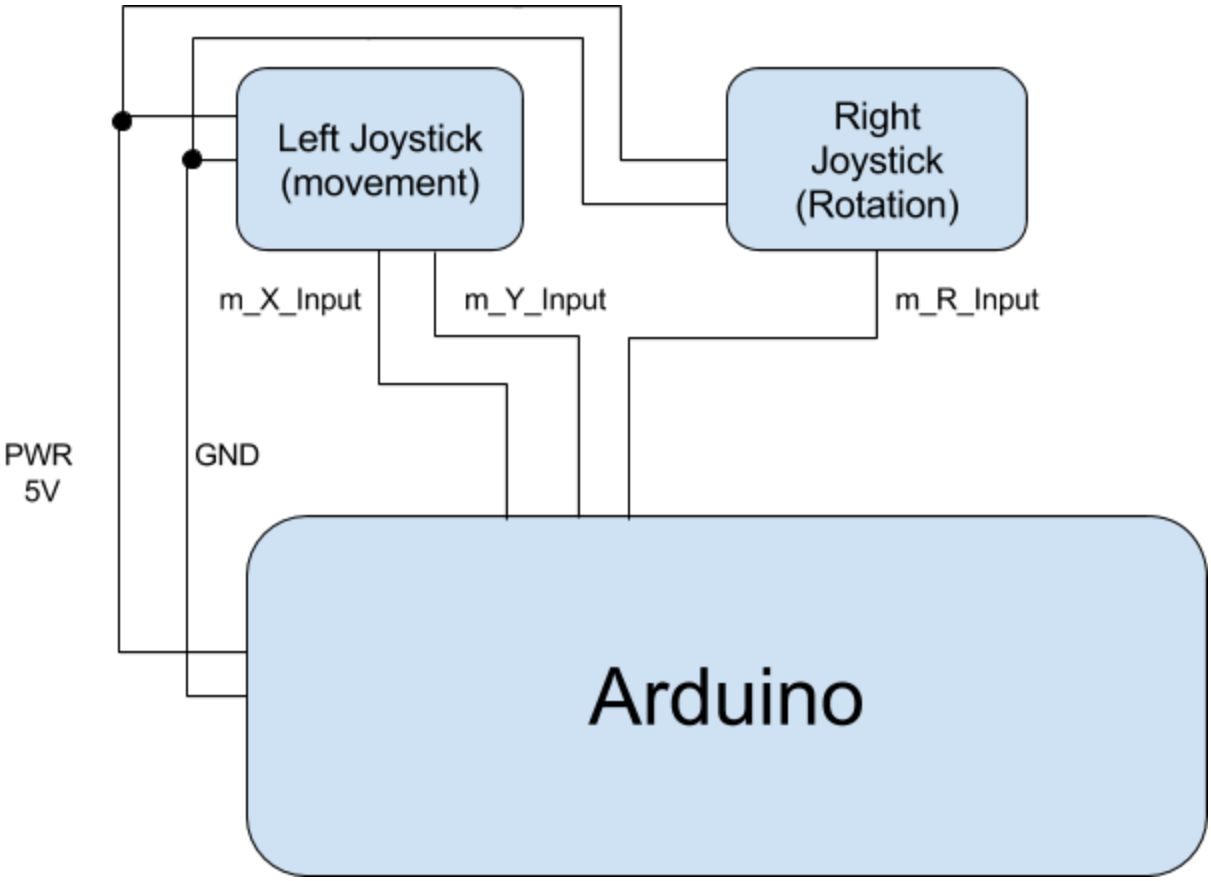
$$Acceleration_{system\ max} = \frac{40.12}{90\ kg\ x\ 0.0762} = 5.85\ {}^{m}/_{s^2}$$

$$\omega = \frac{2\ {}^{m}/_{s}}{0.0762\ m} = 26.25\ {}^{rad}/_{s}$$

$$26.25\ {}^{rad}/_{s}\ x\frac{60}{2\pi} = \mathbf{250\ RPM}$$

11.2.3 Joystick

The joystick control will be responsible for taking the user input and translating it into voltage that the will be sent to the arduino controller. There will be two analog-sticks, one will take in 2-axis input and will represent the forward and lateral movement. The second analog-stick will only take in 1-axis and will represent the rotation. The joysticks will be connected to the 5V and GND pins, as well as one of the analog input pins on the arduino.



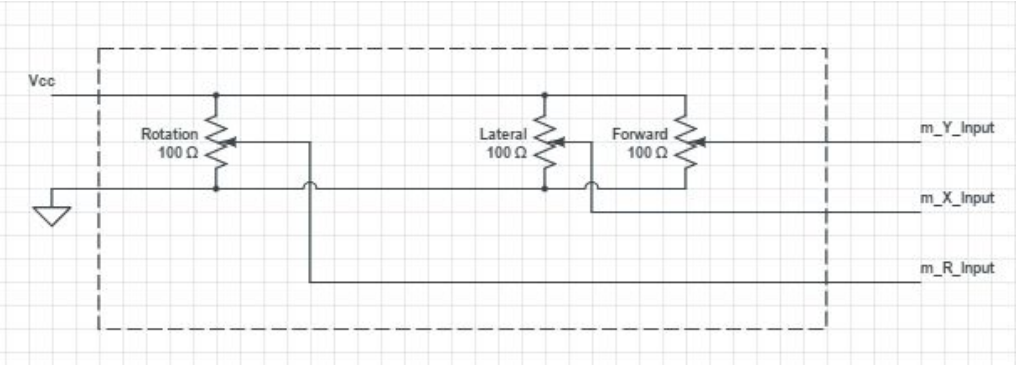


Figure 11: Joystick Circuit

Both the analog-sticks will be attached to a single plastic grip and stand which will house the cable connection to the joysticks. The rotation joystick will be attached to the top of the housing and will be moved by the user's thumb. The second joystick will connect to the bottom of the housing where moving the entire housing will move the joystick input. The grip will be mounted to right the arm of the chair and the cables will go along the chassis to the controller.

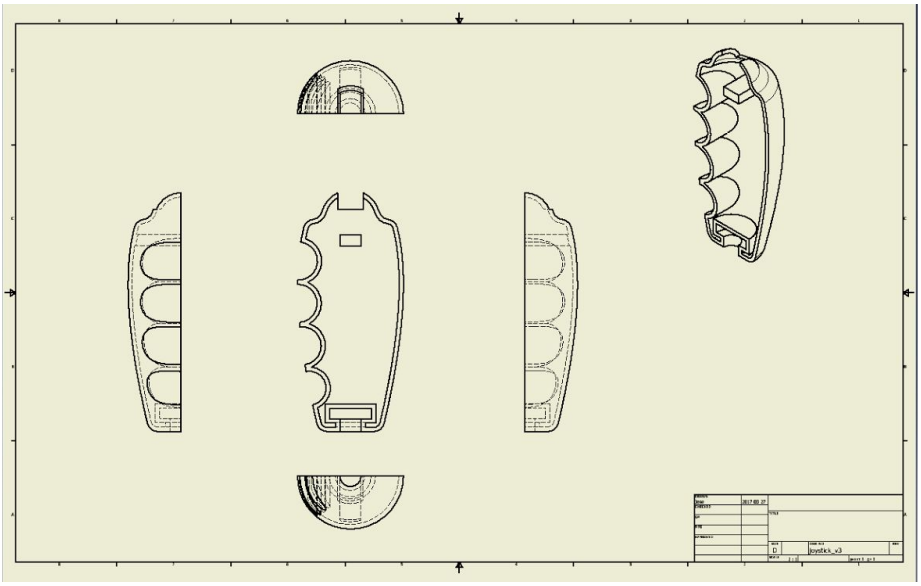


Figure 12: Internal Joystick Grip Mount Model Sheet

11.2.4 Electrical Safety system

In order to ensure all electrical components are protected we have decided to have manual switch and a circuit breaker between the between the battery and all other electrical components. The circuit breaker is comprised of 4 30 Amp fuses, each attached to its own speed controller. The motors have a stall current of 133 Amps, which is well below the fuse breaker requirement and in a case where the system is overloaded or stuck, the motors drawing more current won't be harmful to the system.

Mathematical Justification:
We believe that adding a 30 Amp fuse in series between the battery and the motor controller would greatly protect our motor controllers, but as well battery health in the long run. The particular 30 Amp ATC fuses we have in our system allows for certain amperage spike for a short amount of time which is outlined in the image below. This will allow for short amp spikes that may occur in start up times and operations from 0 speed to moving.

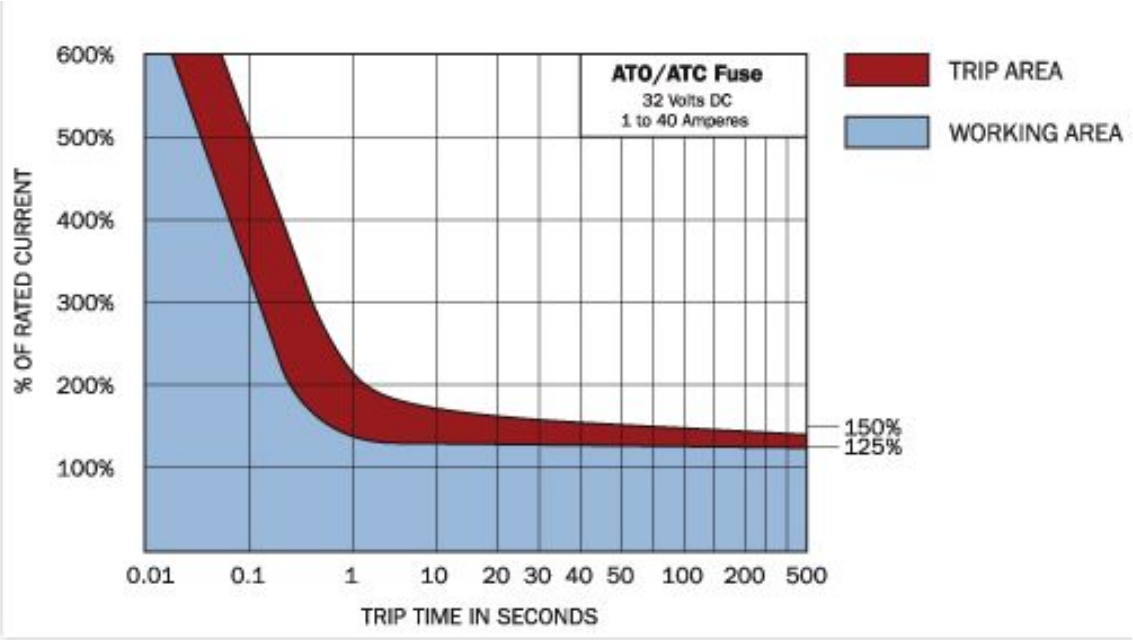


Figure 13: Blow Delay Curve]

Referenced from https://www.bluesea.com/products/5245/ATO___ATC_Fuse_-_30_Amp

11.2.5 Proximity Detection System - Extended Goals

A version 1.0 addition (not in version 0), this would provide added safety measures for the user. The intended function of this is for protection of the user and the system to prevent avoidable collisions with obstacles such that sudden shocks to the system are avoided. The proximity detection system consists of 4 proximity sensors on each side of Dexter such that when within 15 cm of detected obstacle(s), system immediately decelerates to 0.5 m/s and maximum speed is capped at 0.5 m/s. When sensor(s) begin reading values greater than 15 cm, system will reset maximum speed back to 2 m/s.

11.2.6 Battery Low Indicator - Extended Goals

As a protocol for safe practice we have decided to place an indicator for low battery in order to protect the components of the system and the user all the while maintaining high usability of the system. In order to do so we will make a circuit similar to the one shown in figure 5 to indicate to the user that the device is running on low battery and will need to be recharged. “Low” level will be dictated by the zener diode (**ZD)** reverse bias voltage which could be revised accordingly. A red LED will be shown to the user to indicate when the “low” level is met.

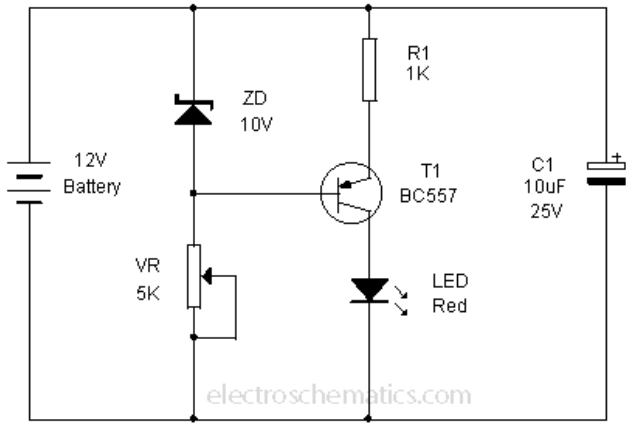


Figure 14: Battery Indicator Circuit [6]

11.3 Software Components

This project requires custom software in addition to hardware design. This is due to the Arduino used to control the mecanum wheels with user input. The software will be written in a C/C++ language subset, compiled by a proprietary compiler designed specifically for the Arduino. The majority of development will be done using the Arduino IDE, with version control being handled by GitHub. it should be noted that currently the exact hardware models to be used has not been decided which means timing constraints have not been finalized for this document. Rough calculations have been done to ensure the maximum required polling rates can be achieved by the Arduino. The basic components which will create the foundation of the software are as follows.

Function Name	Inputs	Outputs	Description
`set_inputs()`	`None`	`None`	Controller method for input value retrieval and processing

`read_inputs()`	Platform Joystick Input	`int valX`
		`int valY`
		`int valR`
			Read and store analog joystick inputs

`shift_inputs()`	`int valX`
	`int valY`
	`int valR`
		`int valX`
		`int valY`
		`int valR`
			Adjust joystick position values to fit calculations

`set_motors()`	`None`	`None`	Controller method for motor value calculations

`initial_outputs()`	`int valX`
	`int valY`
	`int valR`
		`int motors[4]`	Calculates motor values based on joystick input

`scale_outputs()`	`int motors[4]`	`int motors[4]` `int prevMotors[4]`	Limits motor output to within constraints and maps output to values required by speed controllers
`ramp_outputs()`	`int prevMotors[4]`	`int motors[4]`	Controls acceleration by limiting the rate at which the speed controller provides maximum power to the motor
`send_motors()`	None	`int motors[4]`	Send the motor values to their respective speed controllers

Table 9: Function Breakdown

11.3.1 Read User Input

```
read_inputs() {
  // Read joystick inputs
  int x = analogRead(joyX);
  int y = analogRead(joyY);
  int r = analogRead(joyR);

  // Nullify deadzone
  valX = (x >= DEAD_MAX || x <= DEAD_MIN) ? x : HOME;
  valY = (y >= DEAD_MAX || y <= DEAD_MIN) ? y : HOME;
  valR = (r >= DEAD_MAX || r <= DEAD_MIN) ? r : HOME;
}
```

Behavioural Description

This module is responsible for reading the analog input from the joysticks and delivering that information to the processor. While the motion of the right joystick will be physically constrained, the module must also account for a joystick not returning exactly to its home position, hence the implementation of a deadzone surrounding the joystick home position.

Initialization

Initialization of the module will be performed on startup by the Arduino microprocessor, which is responsible for mapping the joystick wires to the correct input pins.

Hardware/Software Decomposition

The input module is responsible for detecting analog input from the controller joysticks and relaying that information to the microprocessor. The joysticks return inputs in a range from 0 to 1024, meaning user input is able to produce gradiated output based on magnitude.

As the inputs returned by each joystick range from 0 to 1024, 512 and upwards is designated as a positive value, and values below 512 are designated as negative values. Due to the unlikeliness of the joysticks returning to their exact home positions, a deadzone of 512 ± 50 is necessary to prevent unintended platform movement while the user expects the platform to be at rest.

Inputs

The inputs to this module are readings taken from the joysticks which control the platform movement

Outputs

The outputs of the module are the processed values of the joystick positions, with each joysticks deadzone accounted for:

- valX - the value of the X position of the left joystick (lateral movement)
- valY - the value of the Y position of the left joystick (forwards and backwards movement)
- valR - the value of the X position of the right joystick (rotational movement)

Constants

- DEAD_MIN - The lower bound of the joystick deadzone
- DEAD_MAX - The upper bound of the joystick deadzone
- HOME - The rest position of the joystick
- joyX - The input pin for the X position value of the left joystick
- joyY - The input pin for the Y position value of the left joystick
- joyR - The input pin for the X position value of the right joystick

11.3.2 Shift User Input

```
shift_inputs() {
  valX = valX - HOME;
  valY = valY - HOME;
  valR = valR - HOME;
}
```

Behavioural Description

This module is responsible for shifting the received joystick inputs from a range of 0 to 1024 to a range of -512 to 512 in order to simplify the calculation of motor values.

Initialization

This module does not require initialization; once the joystick inputs have been received, the module is invoked by the controller method `set_inputs()`

Hardware/Software Decomposition

The module does not interact with hardware.

Inputs

The inputs to this module are the processed values of the joystick positions:

- valX - the value of the X position of the left joystick (lateral movement)
- valY - the value of the Y position of the left joystick (forwards and backwards movement)
- valR - the value of the X position of the right joystick (rotational movement)

Outputs

The outputs of the module are the processed values of the joystick positions, shifted down by 512:

- valX - the value of the X position of the left joystick (lateral movement)
- valY - the value of the Y position of the left joystick (forwards and backwards movement)
- valR - the value of the X position of the right joystick (rotational movement)

Constants

- HOME - The rest position of the joystick

11.3.3 Motor Values Calculation

```
initial_outputs() {
  //logic to set motor value. Adding HOME shifts range back to 0-1024
  motors[0] = valY + valX + valR + HOME; //front left
  motors[1] = valY - valX - valR + HOME; //front right
  motors[2] = valY - valX + valR + HOME; //back left
  motors[3] = valY + valX - valR + HOME; //back right
}
```

Behavioural Description

This module is responsible for calculating the values that are to be sent to each motor's speed controller using the processed joystick input

Initialization

This module does not require initialization; once the joystick inputs have been received, the module is invoked by the controller method `set_motors()`

Hardware/Software Decomposition

The module does not interact with hardware.

Inputs

The inputs to this module are the processed values of the joystick positions:

- valX - the value of the X position of the left joystick (lateral movement)
- valY - the value of the Y position of the left joystick (forwards and backwards movement)
- valR - the value of the X position of the right joystick (rotational movement)

Outputs

The outputs to this module are the calculated values to be sent to the speed controllers:

- motors[4] - An array of 4 values, each value in the array corresponds to the speed value of a certain motor
    - motors[0] - The front-left motor
    - motors[1] - The front-right motor
    - motors[2] - The rear-left motor
    - motors[3] - The real-right motor

Constants

- HOME - The rest position of the joystick

11.3.4 Scale Motor Values

```
scale_outputs() {
  //ensure range of 0 to 1023 then map for servo input range of 0-180
  for (int i = 0; i < 4; i++) {
      ramp_outputs(i);

      if (motors[i] > 1023) {
            motors[i] = 1023;
      }else if (motors[i] < 0){
            motors[i] = 0;
      }

      prevMotors[i] = motors[i];
      motors[i] = (int)map(motors[i], 0, 1023, 0, 180);
  }
}
```

Behavioural Description

This module is responsible for scaling the values that are to be sent to each motor's speed controller. While all calculations were done in the range 0 to 1023, the speed controllers require a value within the range 0 to 180.

Initialization

This module does not require initialization; once the initial motor values have been calculated, the module is invoked by the controller method `set_motors()`

Hardware/Software Decomposition

The module does not interact with hardware, however the method is responsible for mapping the calculated motor values to a range which is compatible with the physical speed controllers.

Inputs

The inputs to this module are the calculated values to be sent to the speed controllers:

- motors[4] - An array of 4 values, each value in the array corresponds to the speed value of a certain motor
    - motors[0] - The front-left motor
    - motors[1] - The front-right motor
    - motors[2] - The rear-left motor
    - motors[3] - The real-right motor

Outputs

The outputs to this module are the calculated values to be sent to the speed controllers, mapped to their final range of 0 to 180:

- motors[4] - An array of 4 values, each value in the array corresponds to the speed value of a certain motor
    - motors[0] - The front-left motor
    - motors[1] - The front-right motor
    - motors[2] - The rear-left motor
    - motors[3] - The real-right motor

Constants

This module does not reference any constants

11.3.2.5 Controlling Motor Acceleration

```
ramp_outputs(int i){
	if(prevMotors[i] - motors[i] != 0){
		//Forwards
		if (prevMotors[i] - motors[i] < 0){
			if(prevMotors[i] + RAMP_INTERVAL < motors[i]){
				motors[i] = prevMotors[i] + RAMP_INTERVAL;
			}
		// Backwards
		} else if (prevMotors[i] - motors[i] > 0){
			if(prevMotors[i] - RAMP_INTERVAL > motors[i]){
				motors[i] = prevMotors[i] - RAMP_INTERVAL;
			}
		}
	}
}
```

Behavioural Description

This module is responsible for limiting the acceleration of each motor in order to mitigate massive increases in speed, especially from a stationary position. The module achieves this by comparing the stored previous motor values to the current values, and adjusts the motor value based on the difference.

Initialization

This module does not require initialization; once the motor values have been calculated, the module is invoked by the method `scale_outputs()`

Hardware/Software Decomposition

The module does not interact with hardware.

Inputs

The input to this module are the current calculated value of a motor and its previous value.

- `motor[i]` - The current value of a motor
- `prevMotor[i]` - The previous value of the motor

Outputs

The output of this module is a calculated value which has been adjusted to limit the speed increase of the motor:

- `motors[i]` - The current value of a motor, adjusted to fall within an acceptable increase interval

Constants

- RAMP_INTERVAL- The maximum allowed difference between the previous motor value and the current motor value

11.3.5 Send Motor Values to Speed Controller

```
send_motors() {
  servo_fl.write(motors[0]);
  servo_fr.write(motors[1]);
  servo_bl.write(motors[2]);
  servo_br.write(motors[3]);
}
```

Behavioural Description

This module handles the sending of the processed motor values to each motors respective speed controller.

Initialization

Initialization of the module will be performed on startup by the Arduino microprocessor, which is responsible for mapping the speed controller wires to the correct output pins.

Hardware/Software Decomposition

The output module is responsible for sending the values of each motor to the speed controllers via the pins each controller is attached to. Each of the motor values lies between 0 and 180, as required by the output library used to send the values to the controllers.

Inputs

The module does not take inputs.

Outputs

- motors[4] - An array of 4 values, each value in the array corresponds to the speed value of a certain motor
    - motors[0] - The front-left motor
    - motors[1] - The front-right motor
    - motors[2] - The rear-left motor
    - motors[3] - The real-right motor

Constants

This module does not reference any constants

12. System Decomposition Tree

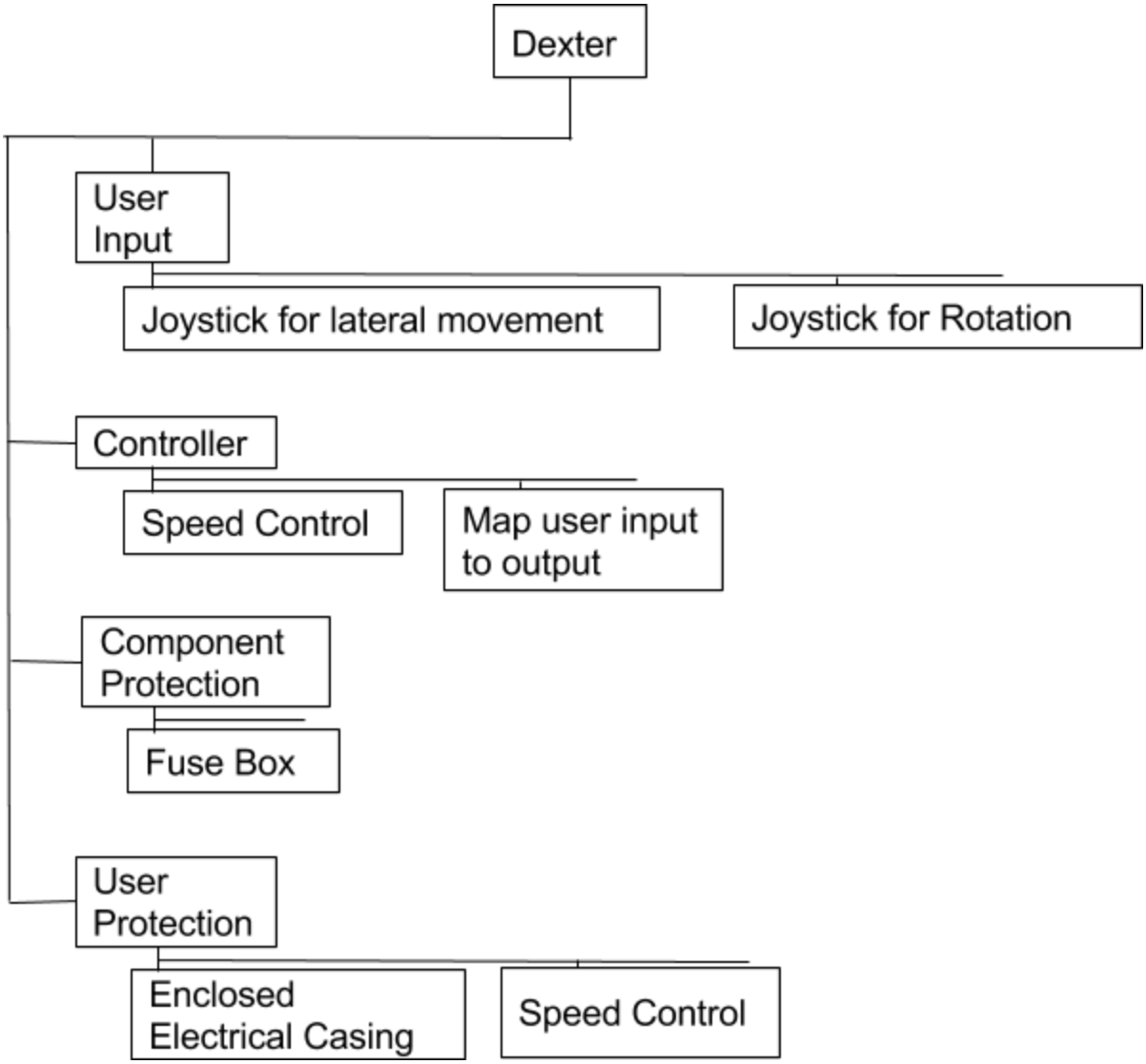


Figure 15: System Decomposition Diagram

13. Design-Requirement Map

Requirement	Design
Support User	Seating and chassis system
Transport User	Mecanum wheels, drivetrain, motor, battery , arduino system
Protect User	Braking system, hardstop coded in software
User Control	Joystick system
Translational Velocity	Mecanum wheel controlled by arduino and motor
Rotational Velocity	Mecanum wheel controlled by arduino and motor
Translational Acceleration	Mecanum wheel controlled by arduino and motor
Angular Acceleration	Mecanum wheel controlled by arduino and motor
Translational Resolution	Arduino sending correct PWM
Angular Resolution	Arduino sending correct PWM
Battery Charge	Battery system

Table 10: Design-Requirement Mapping

14. List of Likely Changes

Section 11.2.5 Proximity Detection System and 11.2.6 Battery Low Indicator were a part of the project's extended goals and due to time and budget constraints they have not been implemented in the system yet.

15. Conclusion

This document outlines the different software and hardware components within our mobility device. During research and conception, the focus was intent on keeping costs at a minimum

while still meeting requirements and goals. During all building and designing phases, safety remained the priority for both the user and the builders.

References

[1]
http://www.tpsgc-pwgsc.gc.ca/biens-property/sngp-npms/bi-rp/tech/accssblt/questions-eng.html
[2]
http://www.andymark.com/NeveRest-20-12V-Gearmotor-p/am-3102.htm
[3]
http://www.vexrobotics.com/mecanum-wheels.html
[4]
http://www.vexrobotics.com/versahubs.html
[5]
https://www.google.com/patents/US20130068543
[6]
http://www.electroschematics.com/5818/battery-self-discharge-indicator/