SMART WALKER SYSTEM DESIGN DOCUMENT

MECHATRONICS 4TB6
GROUP 2

NATHAN FUJIMOTO	1224348
PRAKHAR GARG	1204351
JOSH GILMOUR	1402325
AARON JASS	1218859
TYLER JASS	1218857
FAUZIA KHANUM	1209252
JACK LIU	1215056
GAGAN SINGH	1306242

Table of Contents

Table of Tables

Table of Equations

Table of Figures

Revision History

Rev	Author(s)	Description of Change	Peer Reviewed	Date
-	F.Khanum	Original Document (Copy of System Design Rev 0)	T.Jass	07-Mar-2018
A	T.Jass	Merged in content from Component Design Rev 0 Made several comments and edits based on TA Feedback	T.Jass	12-Mar-2018
B	T.Jass	Finalized Edits and additions	A.Jass	16-Mar-2018

Purpose

This document's purpose is to define and portray the overall system design of all the SmartWalker capstone project. The designs in this document considers all system requirements, defined in the SmartWalker System Requirements document. The document will communicate all the necessary designs for each functional component of the SmartWalker. By the end, the reader should have a clear understanding of what each component is, and how its design meets and satisfies the requirements. The System Design document will be used to begin implementation of the SmartWalker.

Scope

The scope of this project is to design a walker with autonomous abilities to operate within a specific environment. It is assumed the SmartWalker will be operating in a single room with level floors. Most likely it would be a patient's room in a hospital. This document also includes the specification of the system's complete component design. It is intended to give as much detail per component to be able to properly create each component without doubt. All specifications and constraints are labelled clearly for easier reference later when the team reaches the verification and validation stage of the project.
This project will include (in scope) the following items:

- Autonomous driving (see section ***8.1 Autonomous*** for details) between the user and docking station within one indoor room at a time. The docking station is a temporary storage location with charging capabilities.
- Obstacle avoidance in Autonomous mode.
- Motor and brake controls available to the user while the SmartWalker is in Manual mode.

This project will not include (out of scope) the following items:

- Autonomous usage in multiple indoor rooms simultaneously (autonomous movement is limited to one room).
- Global Positioning system (GPS) tracking
- Development of the docking station due to time and budget constraints, however, we will describe its functionality for context and use on its role with the SmartWalker.

The end result of this project will demonstrate the students' cumulative learning through their academic career via a demonstration of the SmartWalker's autonomous ability in a predetermined location. The SmartWalker herein referred to as walker.

1 Assumptions

1. The SmartWalker is designed to be used in a single room (likely the room of a hospital patient)
2. The ceiling and walls are mostly a light shade (beige, light grey, white, etc) and homogenous
3. The floor is expected to be relatively flat (subtle ramps are acceptable but no steps or steep slopes)
4. No clothing, debris, or other possible obstructions are in the way of sensors

2 Context Diagram of Boundaries

The System Context Diagram gives a high-level overview of the information passed to the major system components. The entities and interactions between them are described in ***Table 1 - Entity Descriptions and Interactions.*** More detailed information can be seen in Section ***3 System Data Flow Diagram***.

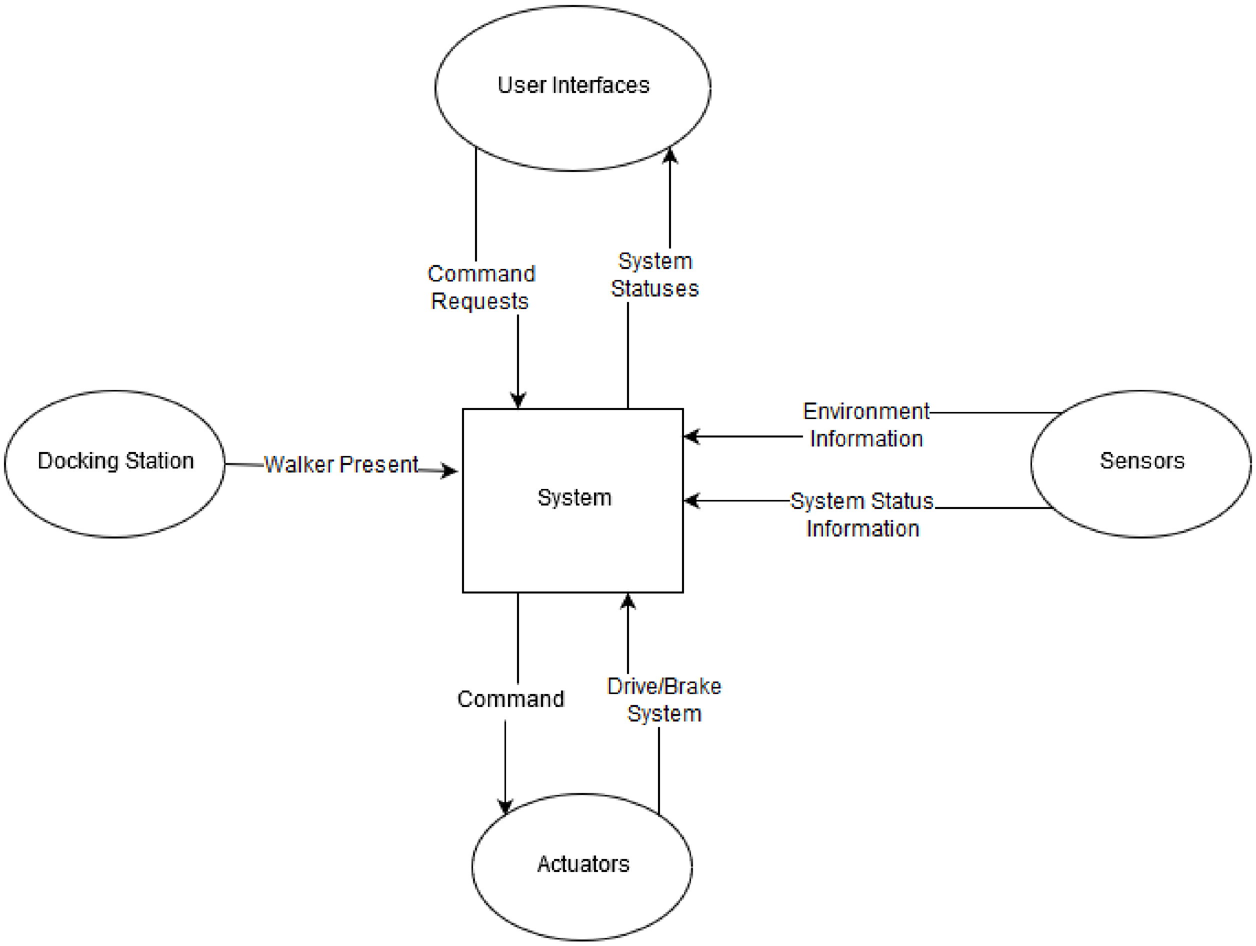

Figure 1 - Context Diagram of Boundaries

Entities	Entity Description	Entity Interactions	Interaction Description
System	The core of all operations. The "system" manages all of the information, computes required information, and outputs information needed by other entities.	*See rows below*	
User Interfaces	These are the components that the user uses to interact with the system, this would include the phone application and the on-board touch screen.	● Request to navigate autonomously (to user/target location/docking station) ● Request to drive manually ● Request to stop moving	● SmartWalker location (at the dock, at the user, in transit) ● SmartWalker behavioural state (idle, navigating autonomously, driving manually)
Docking Station	The docking station is the walker's "Home", it is where it goes to charge.	*The Docking Station functionality is no longer in scope for this project*	
Actuators	Components that execute a physical action. This would include the system's motors.	● Requests to actuate (drive/stop motors)	● Actuator statuses are provided by sensors
Sensors	Components that monitor and measure environment or internal information. This information would include the system's camera vision sensor, proximity sensors, and encoders.	● Requests for SmartWalker environment data (such as object proximity) ● Requests for actuator state/performance (such as motor position)	● Requested environment data ● Requested actuator data

Table 1 - Entity Descriptions and Interactions

3 System Data Flow Diagram

The system Data Flow diagram shows how the different components of the system interact with each other, and with their respective inputs and outputs. The centre of all operations is the Raspberry Pi, which is effectively both the brain and the lesion controlling the flow of all information going in and out of the system.

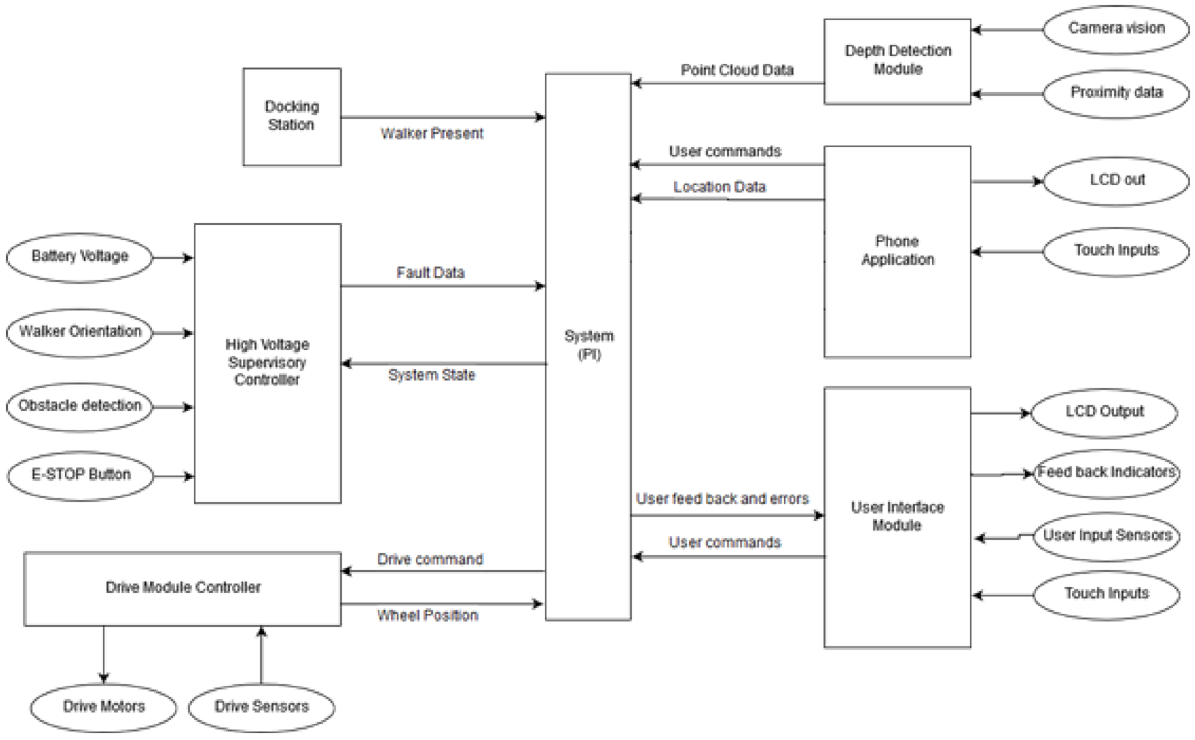

Figure 2 - System Data Flow Diagram

4 Monitored and Controlled Variables

Variable	Description	Units	Range
m_proximity_in_L	Proximity to any objects in the environment (front-left side).	Distance (cm)	[0, 500]
m_proximity_in_R	Proximity to objects in the environment (front-right side).	Distance (cm)	[0, 500]
m_depth_in	Depth of the upcoming ground with respect to the bottom of the walker.	Distance (cm)	[-100, 0]
m_walker_yaw	Orientation of the walker (yaw) with respect to the walker's z-axis.	Angle (deg)	[-180, 180]
m_walker_pitch	Orientation of the walker (pitch) with respect to the walker's y-axis.	Angle (deg)	[-45, 45]
m_walker_loc_out	Global location of the walker.	[Longitude (deg), Latitude (deg)]	[-180, 180; -90,90]
m_walker_loc_in	Location of the walker inside the patient room, with respect to a preset coordinate system.	[x,y] on room map	[0, 100; 0, 100]
m_human_loc_in	Location of the human based on their phone, with respect to a preset coordinate system.	[x, y] on room map	[0, 100; 0, 100]
m_human_yaw	Orientation of the human (yaw) about the human's z-axis.	Angle (deg)	[-45, 45]

m_desired_loc	User selected desired location for the walker in autonomous mode	Enum (states)	GoDocking, GoHuman
m_dock_aligned	Walker is physically aligned with the docking station	Boolean	0, 1
m_user_in_L	Desired forward assisted movement for the user, for the left motor.	Custom	[-100,100]
m_user_in_R	Desired forward assisted movement for the user, for the left motor.	Custom	[-100,100]
m_emergency_signal	Signal for monitoring an emergency situation.	Boolean	0, 1

Table 2 - Monitored Variables

Variable	Description	Units	Range
c_motorL_speed	Left motor speed.	Angular velocity (deg/s)	[0, 50]
c_motorR_speed	Right motor speed.	Angular velocity (deg/s)	[0, 50]
c_brakeL_position	Left brake position.	Brake Position (%)	[0, 100]
c_brakeR_position	Right brake position.	Brake Position (%)	[0, 100]
c_charge_walker	Charge the walker.	Boolean	0, 1
c_dock_connected	Walker physically connected to the docking station.	Boolean	0, 1

Table 3 - Controlled Variables

5 Constants

Number	Description	Value
1	Number of Kinect Sensors	1
2	Location and orientation of Kinect sensors	***Refer to Figure 3*** - Kinect and Ultrasonic Orientation
3	Number of ultrasound sensors	4
4	Location and orientation of ultrasound sensors	***Refer to Figure 3*** - Kinect and Ultrasonic Orientation
5	PID Coefficients	(Kp, Ki, Kd) = To be determined
6	SmartWalker Input Voltage	12 Volts
7	Number of Arduinos	3
8	Number of Raspberry Pis	2
9	Number of Motor Actuated Wheels	2
10	Number of Motor Control boards	1
11	Number of encoders (quadrature)	2
12	SmartWalker maximum speed when in autonomous mode	1 meters/second
13	SmartWalker maximum speed in manual mode	1 meters/second ???
14	Minimum allowable distance to objects in the environment during autonomous navigation	0.25 meters
15	Distance from target position the SmartWalker should stop at after autonomous navigation	0.25 meters

Table 4 - Constants

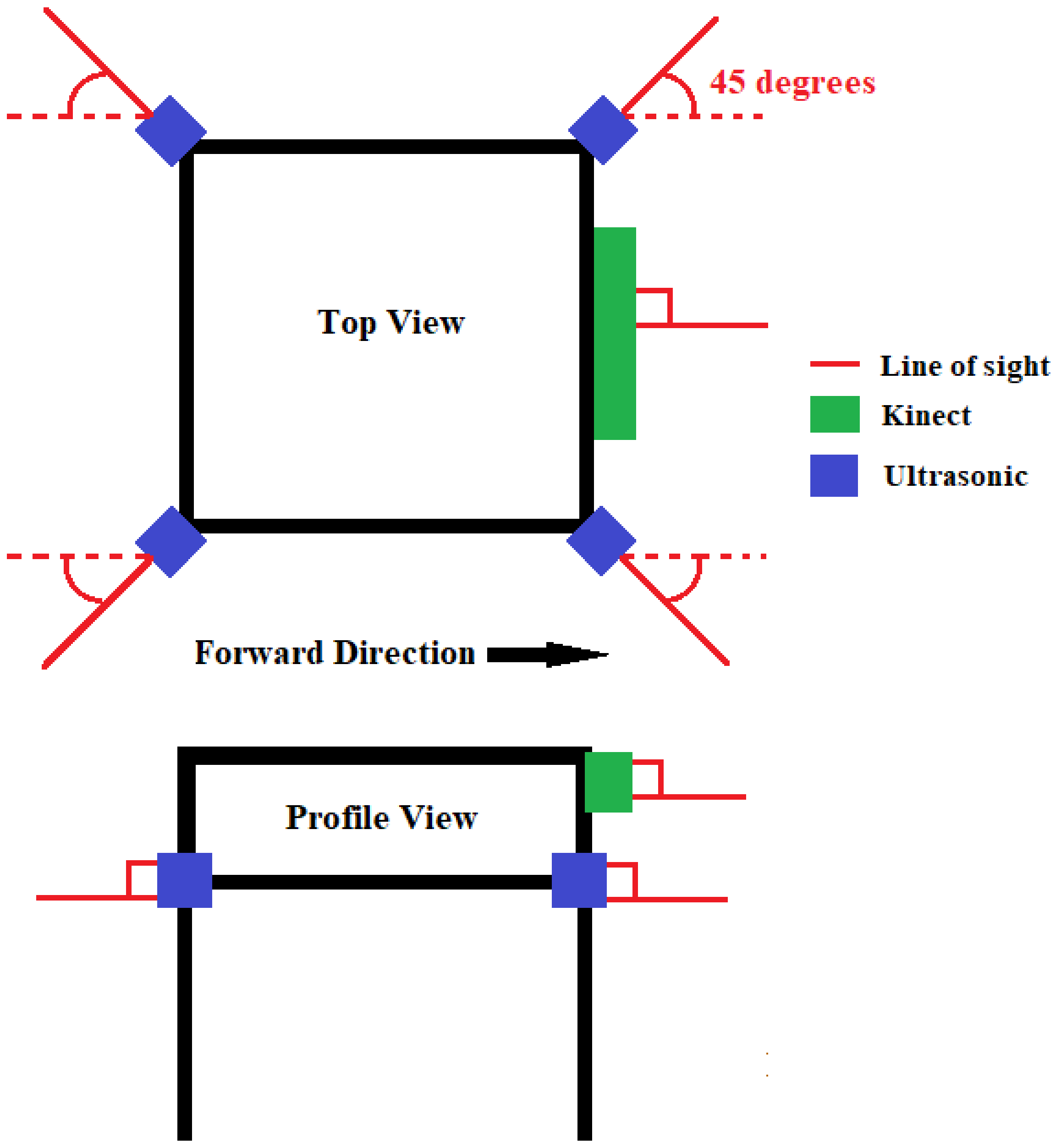

Figure 3 - Kinect and Ultrasonic Orientation

6 Behaviour Overview

6.1 Overall System Behaviour

The behaviour of the walker system can be seen by the following behavioural state diagrams. To describe at the main behavioural components of the walker system, three state diagrams were outlined: the overall system behaviour diagram, the autonomous behaviour diagram, and the manual mode behaviour diagram. The latter two main sub states in which the walker primarily operates in. In any state, the walker is perceiving information from the environment for both functioning and state transitioning.

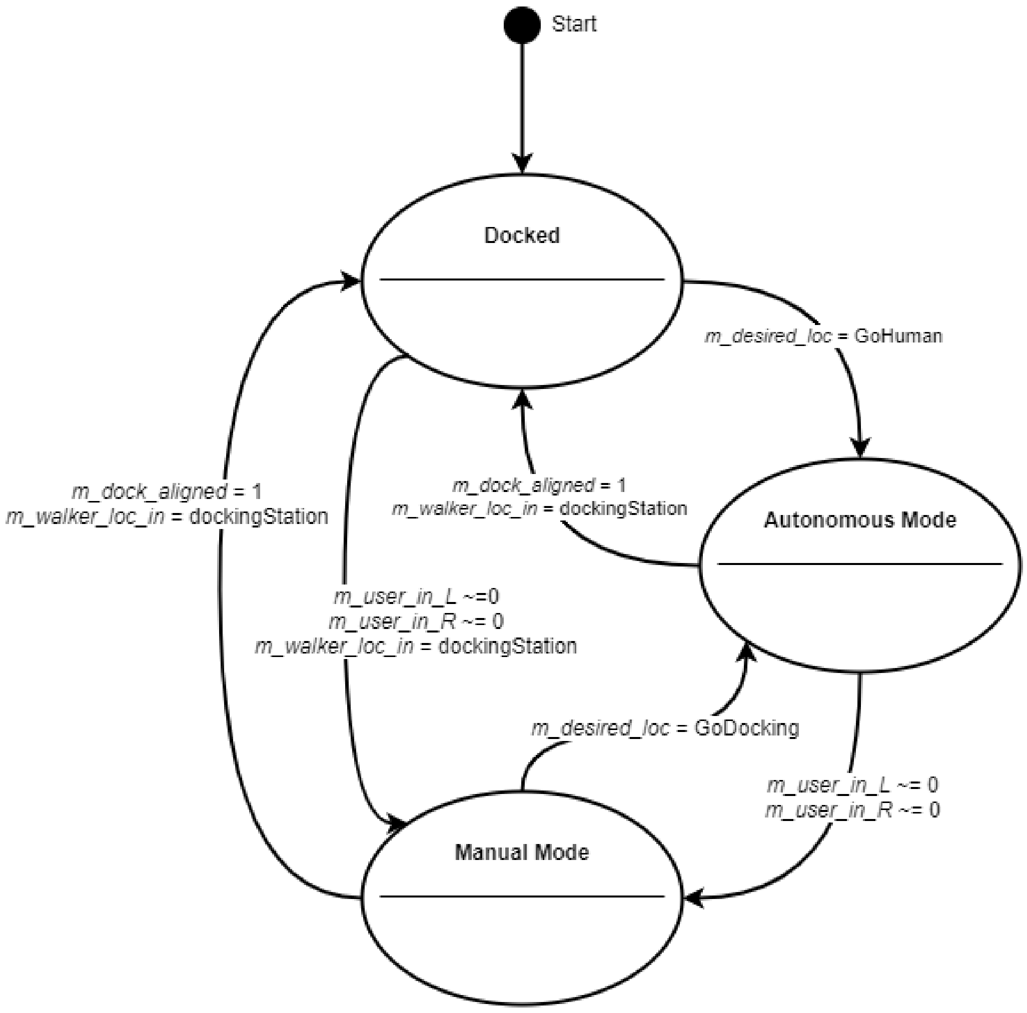

Figure 4 - Overall System Behaviour Diagram

State	**Description**
Docked	In this state the walker is physically connected to the docking station. When connected, the walker is charging its onboard power supply.
Autonomous Mode	In this state the walker is autonomously navigating to and from the user and the docking station (or its current position).
Manual Mode	In this state the walker is being manually operated by the user. This is intended as standard walker usage, with additional safety features.

Table 5 - State Table

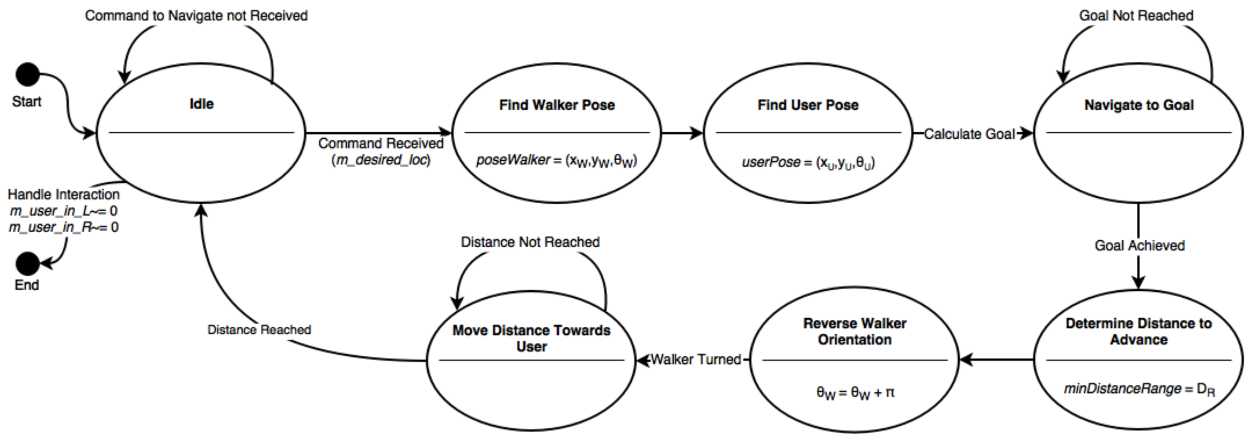

Figure 5 - Autonomous Mode Behaviour Diagram

Please note, whenever the walker is put into an unintended event, a safety check will be done of the status of the system, and the status will be fixed if possible and sent back to the idle position.

State	Description
Idle	In this state the walker is in a waiting phase where it is waiting on instructions on where to navigate by the user, via the user interface.
Find Walker Pose	In this state the walker determines its relative position on the constructed map grid, as well as its relative yaw angle on the map.
Find User Pose	In this state the walker system determines the relative position of the user on the constructed map grid, as well as the user's relative yaw angle on the map.
Navigate to Goal	In this state, considering the walker pose and user pose, the walker calculates the navigation goal and thus begins to navigate to the location.
Determine Distance to Advance	In this state the walker determines how far it can safely approach the destination.
Reverse Walker Orientation	In this state the walker orientates its yaw relative yaw angle so that its handles are facing the user (180-degree turn).
Move Distance Towards User	In this state the walker creeps until it has reached the safe approaching distance. The system then goes back to the idle state.

Table 6 - Autonomous Mode States

As described in the overall system behaviour diagram, if either the walker has reached the docking station or the user physically interacts with walker will the behaviour exit the autonomous mode.

6.2 Navigation Goal Calculation

The walker and user pose will both be determined by the beacons in the room. The goal is the location where the robot is supposed to travel based on where the user is. The goal is calculated to be some distance, K, in front of the user. It is calculated as:

$$Navigation\ goal = \{X_u + \mathrm{K} * \cos\theta_u\,, Y_u + \mathrm{K} * \sin\theta_u\,, \theta_u + \pi\}$$

Equation 1 - Navigation Goal Calculation

The pi added to the user orientation ensures that the walker is facing the user at the end of navigation. This facilitates the measurement of distance between the walker and user in the next state. The walker is then to turn around and approach the user to a reasonable distance (to be determined).

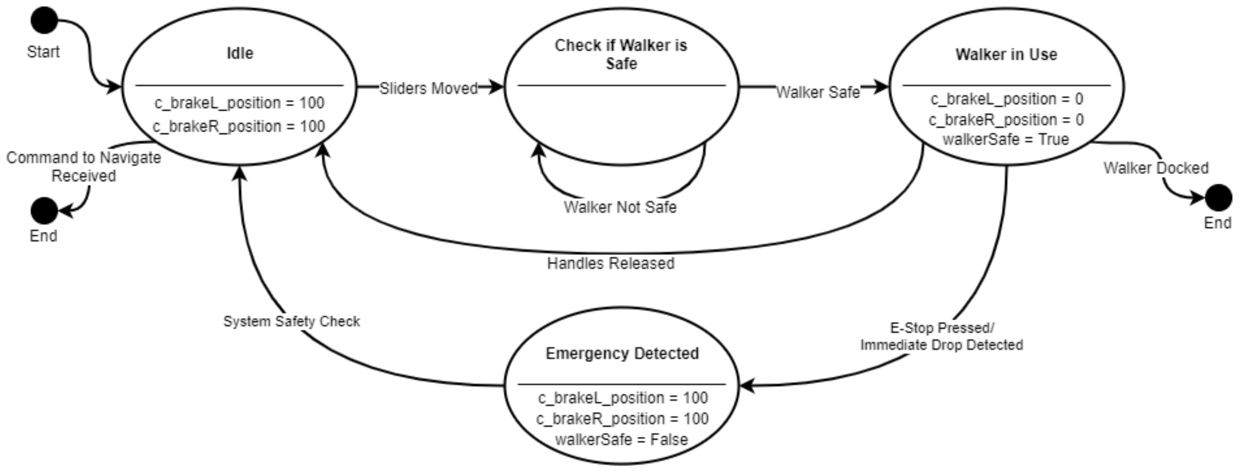

Figure 6 - Manual Mode Behaviour Diagram

State	Description
Idle	In this state the walker is in a waiting phase where it awaits physical interaction from the user, in the form of the handle sliders being moved.
Check if Walker is Safe	In this state the walker immediately checks the environment if there are any detectable dangerous obstacles or landmarks in the vicinity of the walker.
Walker in Use	Once the system deems it is safe, in this state the walker operates like a traditional walker, where it helps the user move him or herself.
Emergency Detected	In this state an emergency was detected and the walker brakes so it cannot be further moved. A safety check is completed so the emergency situation is no longer present.

Table 7 - Manual Mode States

Please note, the Emergency Detected includes the detection of any hazardous situations the user is in the vicinity of (oncoming drops, large slopes, etc..). The safety check will enact functionality to aid the user if wished to proceed through the hazard, if possible (please refer to Section ***8 Normal Operation***) Manual mode behaviour is left when the user asks for the walker to navigate back to the docking station, or to a new position where the user would be.

7 Component Details

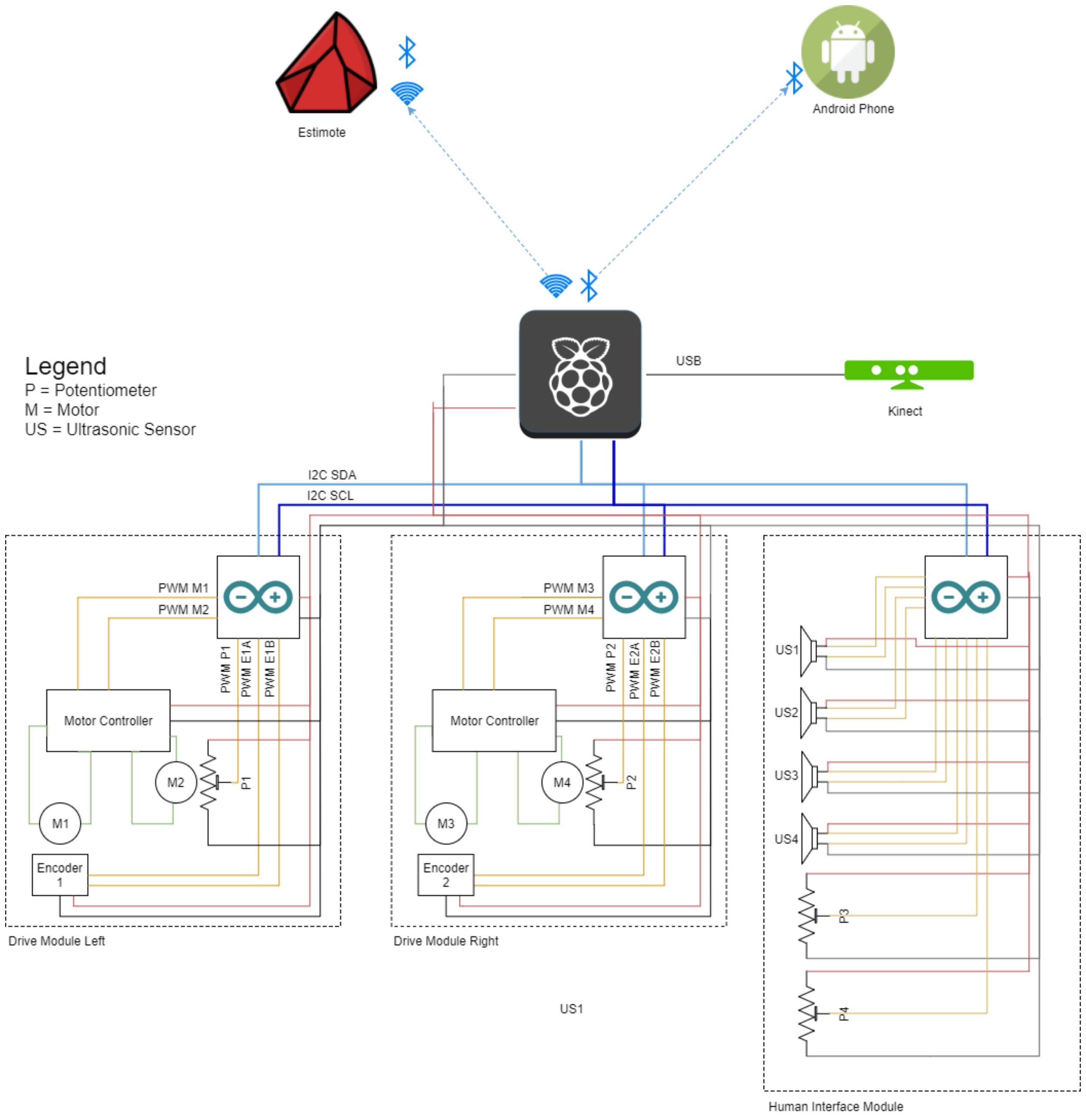

Figure 7 - Electrical Component Diagram

Component	**Inputs**	**Outputs**	**Description**
Raspberry Pi	Modules: ● Microsoft Kinect ● Estimotes ● Android Mobile Phone	Modules: ● Drive Module Left ● Drive Module Right ● Human Interface Module	The main supervisory module in the walker System. It directs data to and from the other modules to assure functionality.
Estimotes	Monitored Variables: ● *m_walker_loc_in*	Modules: ● Raspberry Pi	Wireless beacons to be used for indoor location and navigation of the walker System.
Android Phone	Monitored Variables: ● *m_human_loc_in* ● *m_human_yaw* ● *m_desired_loc*	Modules: ● Raspberry Pi	The assumed personal electronic device that is owned by the user. This device will host the walker Application that will transmit commands and user location information, and receive system system status information.
Kinect (Sensory Module)	Monitored Variables: ● *m_proximity_in_L* ● *m_proximity_in_R* ● *m_depth_in* ● *m_walker_yaw* ● *m_walker_pitch*	Modules: ● Raspberry Pi	Sensor returning point cloud data about the environment. This sensor is vital to the system's object detection.
Drive Modules	Modules: ● Raspberry Pi ● Human Interface Module	Control Variables: ● *c_motorL_speed* ● *c_motorR_speed* ● *c_brakeL_position* ● *c_brakeR_positio n*	Modules that will do computations on the internal system data and external environment data from the sensors and use it to control the actuators (motors) appropriately.
Human Interface	Monitored Variables: ● *m_user_in_L*	Modules: ● Drive Modules	This module is the physical interface between the

Module	● *m_user_in_R* ● *m_emergency_signal* Modules: ● Raspberry Pi		drive module and the user. It allows the user to manually drive the walker system's motors.

Table 8 - Component Description

Module	**Derive Timing Constraints**
Kinect	Cannot read data from the Kinect faster than the Raspberry Pi can process it whilst also maintaining other system functionality
Android Phone	Will communicate messages from the user at the standard Bluetooth 4.0 speeds to the Raspberry Pi (on the walker), and back to the user.
Estimotes	The beacon responsiveness will adhere to the Raspberry Pi's scanning of the beacons while in Autonomous Mode.
Raspberry Pi	Must communicate with the arduinos at a rate they can handle and are configured to accept.
Drive Modules	Arduinos must communicate with the Pi and motor controllers at a rate they can handle and are configured to accept. The arduinos must run fast enough that it does not miss any encoder counts and are able to keep up with the rate of them.
Human Interface Module	The status of the human and environment interactions must be checked often enough that no user interactions are missed. Braking inputs and ultrasonic data will adhere to serial 'I-squared-C' speeds for sending data to the Raspberry Pi and to the Drive Modules.

Table 9 - Derived Timing Constraints

Module	**Initialization**
Kinect	Run the libfreenect driver to allow reading data from the Kinect.
Android Phone	The app must be running on the phone and a bluetooth connection established with the walker before several walker features can be used, such as autonomous navigation.
Estimotes	Physically the beacons required to be on the walls of the room, at the centre of each wall. Additionally, one would have to be inside the walker and connected to the Raspberry Pi at initialization.
Raspberry Pi	The Raspberry Pi will be turned on and have the necessary software pre-installed.
Drive Modules	The Arduinos, motor controllers, motors, and encoders will all have to be turned on via power-up. The arduinos will have the necessary software pre-installed.
Human Interface Module	The arduino and the sensors to interact with the human and environment (brakes and ultrasonic) will be initialized at power-up.

Table 10 - Initialization

8 Normal Operation

The following section will describe the different modes and states each mode has in more detail by stepping through possible positive normal scenarios.

Please note since this is the Draft System Design that we have not determined all details yet and have noted said details to be defined later.

8.1 Autonomous

As mentioned in our System Requirements document, autonomous mode is the ability to move out to a designated home location as well as return to the user when requested. Autonomous mode consists 3 states: moving to user, moving to docking station, and docked. The following sections will describe positive normal scenarios (behaviour the system should cover and everything proceeds as expected).

8.1.1 Moving to user

1. User is not detected by handlebar sensors (m_user_in_L and m_user_in_R)
2. User requests the walker to come to user via phone app
3. Phone app sends bluetooth signal to walker
4. If walker is not within bluetooth distance?
    a. Display message to user “Walker out of reach.”
5. If walker is within bluetooth distance
    a. Walker retrieves walker pose from estimotes
    b. Walker retrieves user pose from estimotes
    c. Walker calculates the goal pose and begins navigating to it, as described before
    d. Once the walker reaches the goal, the distance to the user is measured
    e. The walker turns 180 degrees
    f. The walker backs up to within a reasonable distance from the user
    g. Walker returns to idle state

8.1.2 Moving to docking station

1. User is not detected by handlebar sensors (m_user_in_L and m_user_in_R)
2. User sends signal to dock via phone app
3. If walker is not within bluetooth distance
    a. Display message to user “Walker out of reach.”
4. If walker is within bluetooth distance
    h. Walker retrieves walker pose from estimotes
    i. Walker retrieves docking station pose from estimotes
    j. Walker moves to docking station
    k. Walker docks into charging port (m_dock_aligned)
        i. Go to Docked state

8.1.3 Docked

1. Walker is at the docking station and connected (c_dock_connected)
    a. If battery is less than 100% (m_battery_lvl)
        1. charge (c_charge_walker*)*
    ii. If battery is at 100% (m_battery_lvl)
        1. Disengage charging and go to Idle state

8.2 Manual

As mentioned in our System Requirements document, Manual mode is the ability to operate like a standard medical walker, with added smart features such as assisted braking and assisted driving. Manual mode has 4 states: regular walking, slope change, objects detected, and idle. The following sections will describe positive normal scenarios (behaviour the system should cover and everything proceeds as expected).

8.2.1 Regular Walking

This state is when the user is utilizing the walker as intended; aiding in regular walking.

1. User detected at handlebar sensors (m_user_in_L and m_user_in_R)
    a. Detect if the walker is moving and surrounding environment
        i. Slope change → Slope Change mode
        ii. Objects in way → Objects detected
        iii. User is utilizing walker for normal walking
    b. User applies brakes manually (c_brakeL_position and c_brakeR_position)
        i. reduce speed accordingly
2. If user is not detected (for more than a certain amount of time, to be determined)
    a. Enter idle state

8.2.2 Slope Change

This state is for when the user is detected at the handlebar sensors, the user is walking around with the aid of the walker and the smartwalker has detected a change in slope of the ground.

1. Query the slope of the ground using the onboard sensors
2. Calculate the pitch of the walker (m_walker_pitch)
3. If the pitch of the walker is positive or negative 5 deg to a certain deg
    a. This state has yet to be finalized, more details to come in the future

8.2.3 Idle

1. User is not detected at handlebar sensors (m_user_in_L and m_user_in_R)
    a. If walker is not detected at docking station (m_dock_aligned)

  - Walker waits for command from user

b. If walker is detected at docking station (m_dock_aligned)
  - If battery is less than 100% (m_battery_lvl)
    - charge walker (c_charge_walker)
  - If battery is 100% (m_battery_lvl)
    - Sit idle (details of idle mode to be determined)

9 Undesired Event Handling

Section 7 covered normal operation of the walker, this section will talk about abnormal scenarios and how the walker will handle them. Each section will describe the undesired event and the steps the system will take to ensure that the event is resolved in a safe manner.

9.1 Emergency Stop

Emergency stop refers to when the user pushes/applies the emergency stop or the system determines that there is an emergency and sends a signal at any point while the walker is in normal operation. This option is for when the user

1. Emergency Stop activated
    a. Brakes applied to motors in a safe manner (c_brakeL_position and c_brakeR_position)
    b. State of emergency (walkerSafe = False)
    c. Display prompt for user to cancel emergency state
    d. Wait for user input
        i. Safe to continue OR enter manual mode (cancel initial request)

9.2 User Detected During Autonomous Mode

The following scenario is for the walker is in autonomous mode and a user is detected by the handlebar sensors.

1. Walker is in autonomous mode
2. User is detected by handlebar sensors (m_user_in_L and m_user_in_R)
    a. Brakes applied to motors in a safe manner (c_brakeL_position and c_brakeR_position)
    b. Walker enters manual mode

9.3 Object Collision with Smartwalker

The following scenario is for when the smartwalker is in autonomous mode and has collided with an object of a minimum size yet to be determined.

1. Walker location does not match expected location
    a. Brakes applied to motors in a safe manner (c_brakeL_position and c_brakeR_position)
    b. State of emergency (walkerSafe = False)
    c. Wait for user input
        i. Safe to continue OR enter manual mode (cancel initial request)

9.4 Walker has Fallen Over or Very high Incline/Decline

This state is when the walker has fallen over or is at a dangerously high incline or decline. This is a very undesired and not recommended.

1. Walker gyroscopes detect angle greater than 15°
    a. Brakes applied to motors in a safe manner (c_brakeL_position and c_brakeR_position)

   b. State of emergency (walkerSafe = False)
   c. Wait for user input
      i. Safe to continue OR enter manual mode (cancel initial request)

9.5 Goal location is unreachable

When the walker is in autonomous mode, it has a goal of either the user or the docking station. In moments when the path is not clear to the destination, the following will occur:

1. Walker attempts to reach destination
2. Several attempts at reaching goal occur (attempt number and duration to be determined)
3. Walker stops attempting to reach goal
4. Go into idle mode
5. Message is sent to the user’s phone “unable to reach destination.”

9.6 Walker cannot “see”

The following state refers to when the walker sensors are not functioning properly either due to sensor malfunction or a blockage (items covering the sensors) and therefore reporting false data. The walker is in an environment it is not capable of reliably navigation through and thus navigation will cease autonomous mode and the walker will return to manual mode. The user is then able to move the walker to a safe location for troubleshooting.

9.7 Walker is unable to move while autonomously navigating

When the walker is in Autonomous mode and Autonomous navigation with cease and the walker will return to idle mode. The user will be notified via the phone application of the problem.

9.8 Extremely low battery

Since the walker is very dependent on power when the battery level is very low (low level to be determined) the walker will the take the following actions.

1. User will be notified of the low battery state via the phone application
2. The walker will shut down to avoid unexpected behaviour at low power and to avoid damaging the battery

10 System Overview

The system is divided into 3 main components, mechanical, electrical, and software, which are then further divided into subcomponents. Each component and subcomponent behaviours and relationships with one other are also defined here.

The following diagram depicts a simple breakdown of the SmarWalker system along with the inputs from the environment and the outputs to the actuators.

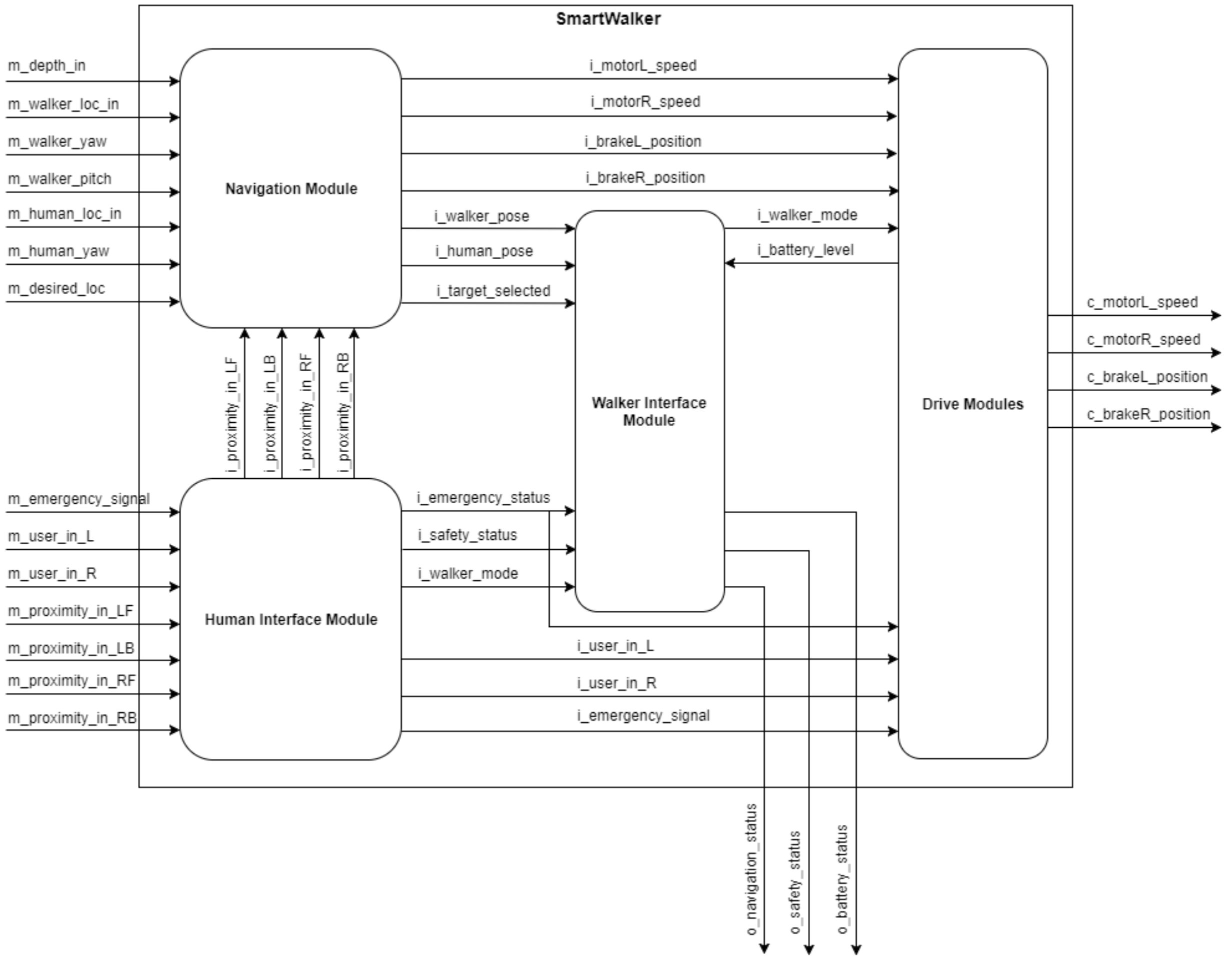

Figure 8 - System Component Breakdown

11 Mechanical Components

The mechanical components of the system consist of only a few parts. The walker, which was purchased whole, for the purpose of being augmented for this project. The remaining portions of the mechanical system comprises of motorized wheels, specialized handles, and mounts for the sensors and other electronic parts.

11.1 Walker

A simple 2-wheel walker was chosen as the basis for the SmartWalker system. It was cheap, sturdy, and simple enough that modifications would be not be complicated. It is 33.81 inches in height, 23.85 inches wide, and 21.41 inches in length. The walker will be refitted with numerous sensor mounts on its frame as well as motorized wheels.

The motorized wheels will be added to the back of the walker, closer to the user, for easy turning. The 2 wheels that came with the walker will be removed and replaced with the skids. The wheels have a built-in torque sensitive clutch to prevent the walker from pulling the human with too much force. The maximum force exerted by the wheel is determined by the compression of an internal spring that is determined by a secondary motor. The position of the secondary motor is based on environmental conditions and will be determined by the control systems described in the Electrical and Software components section below.

All additions to the walker are designed to be easy to mount and reposition on the walker as they do not depend on pre-existing holes or extensive amount of modification to the original walker. Sensors can be easily re positioned or re oriented as required. The wheels are designed to be mounted on the back of the walker however can be moved to the front to change the driving experience of the walker.

An electronics box will be mounted on the middle horizontal border of the walker and will double as a seat for the user. The main purpose for this is to shield the electronic form the external environment while hiding the bulkier components in a visually appealing way. The electrical box should be the key location to troubleshoot all electrical problems associated with the walker. As the electrical box/seat is still under development, its designs have not been displayed Appendix II.

11.2 Wheels

The internal components of the wheel are designed to change the torque exerted by the wheel based on the environments. The wheel is composed of two major components. The first is the internal clutch assembly which determined the transfer of torque. The second major component is the wheel itself which is mounted around the clutch assembly.

This clutch system is designed to make the walker safe to use by preventing the walker from pulling the user with too much force. All the components of the clutch are implemented within the wheel assembly to reduce weight and prevent the additional components from making the walker difficult to use. The clutch assembly (inside of the wheel) design is based on the mechanisms inside a drill clutch. It is composed of a variable position plate controlled by a secondary motor which determines the compression of an internal spring which forces a plate. the pressure on the plate determines the amount of force transferred through the bearing balls which in turn drive the wheel hub.

The wheel itself is a 6061-aluminum drum mounted on two custom bearings around the clutch housing. The drum bearing assembly prevents the need to balance the wheel while providing a significant amount of space inside. Furthermore, the drum assembly is very strong and can withstand the weight of a user with negligible deformation. CAD drawings for the wheel assembly are available in Appendix 2 Drawings 6 and 7.

11.3 Handles

The handles of the walker will be modified to include a linear potentiometer that will allow the system sense whether or not a user is present at the SmartWalker. Sliding handles instead of gripping handles were chosen when considering our target user. The target user is most likely ill or injured and has the possibility of weak or injured hands or forearms. In such cases, grip handles would be too difficult to operate.

The handles position will determine whether the walker accelerates, decelerates, or brakes. There will also be a toggle button that will determine whether the walker is in “free roll” (brakes disengaged) or not.

11.4 Sensor Mount

A key component of the walker’s controls system are its sensors. It is essential for the sensors to be mounted at the correct positions and orientations to allow the software to acquire accurate data about the physical environment and make well informed decisions. As the control systems of the walker are still being developed the positions of the Kinect and ultrasonic sensors may need to be constantly moved and reoriented. All the mounts are designed to clamp onto the walker frame without requiring mounting holes. This allows the sensors to be mounted and moved with the use of only a single bolt.

12 Electrical Components

The electronic components of our system consist of a low voltage circuit and high voltage circuit. The low voltage circuit handles all communication between the various microprocessor boards as well as all sensor inputs/outputs. The high voltage circuit is utilized to power the drive motors as well as the microprocessor boards. The following sections will describe the list of materials involved in each circuit, its purpose and a schematic diagram of the circuit itself. For a list of parts and their specifications, please refer to Appendix I.

12.1 Low Voltage Circuit

A low voltage circuit was designed so that each of the each of the main controllers and sensor units within the SmartWalker system would be able to communicate with each other without drawing too much current and draining the battery. The connections between each were implemented to meet the hardware specifications of each unit, while also ensuring the safest implementation for each.

The three main units involved in the low voltage circuitry are:

1. Communications between the raspberry pi controller, three Arduino controllers, and the motor controller boards.
2. Sensor communication between the Raspberry pi, Estimotes mapping beacons, android smartphone, touchscreen user interface, and Kinect sensor.
3. Sensor communication between the Arduino board and the four ultrasonic sensors, two potentiometers and two encoders.

All the hardware units are shown in ***Figure 9 - Low Voltage Circuit***, with descriptions and specifications of each.

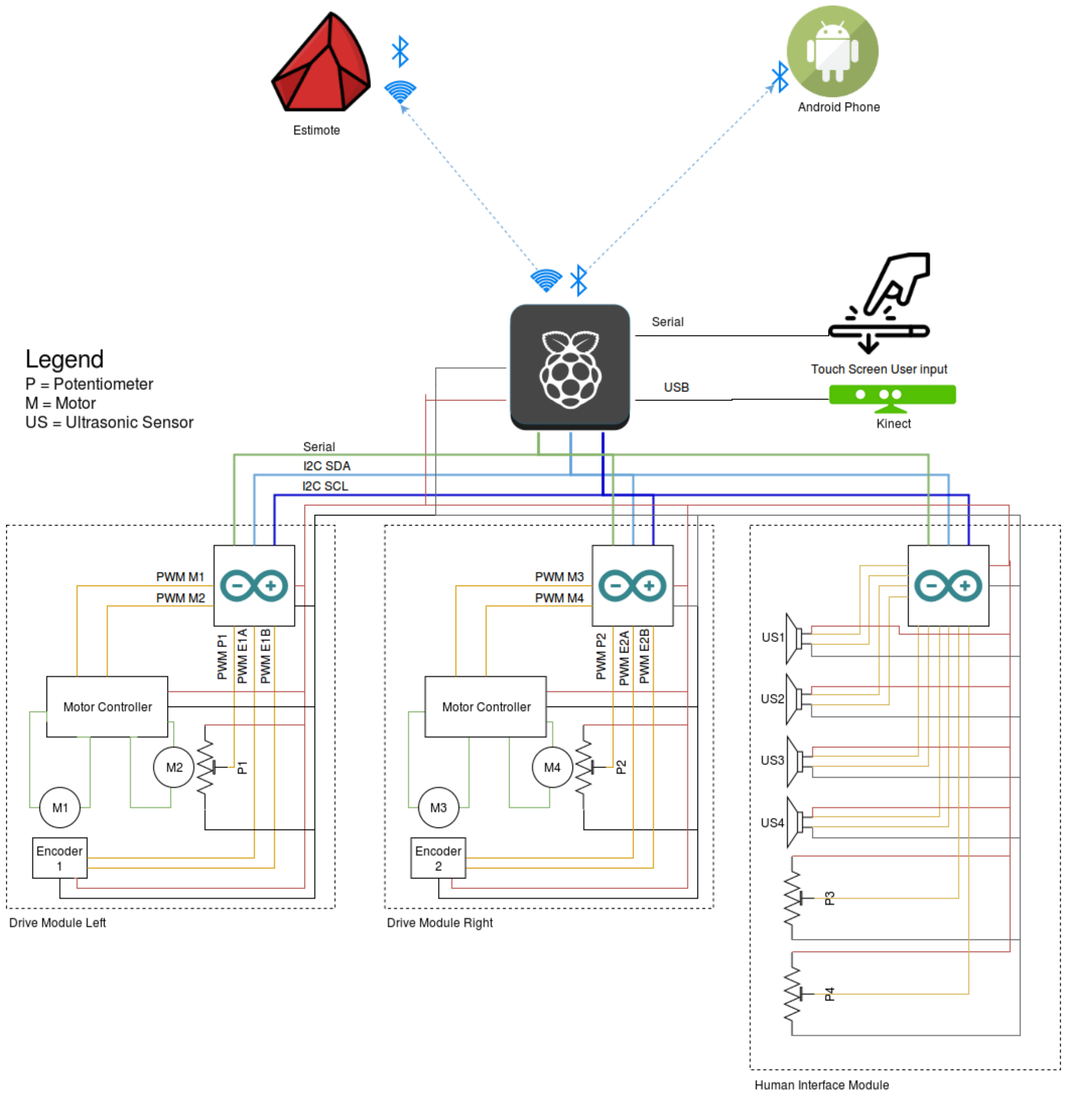

Figure 9 - Low Voltage Circuit

12.1.1 Kinect

The Kinect is a depth camera that the system uses to obtain point cloud data. This data is translated into what is effectively a laser scan for use by our software modules. The point-cloud data collected by the Kinect will be communicated over a serial line and processed by the Raspberry Pi. The Kinect was selected because there are various online resources providing easy integration with ROS, the software being used for most of the system's navigation control.

12.1.2 Estimotes

The Estimotes are beacon location sensors to be used for the indoor positioning of the SmartWalker system, the docking station, and the users phone. These positions are essential to the system's navigation. The Estimotes will communicate a coordinate position and rotational orientation (in the horizontal plane) over WIFI to the Raspberry Pi. These were selected because of the online resources providing easy integration with the Raspberry Pi and ROS libraries.

12.1.3 Motor Controller board

The Motor Controller boards communicate with the Arduino using a low voltage PWM signal. Once the signal is received, the board uses its onboard circuitry to drive the motors. The controller board was selected because of its unique dual channel operating mode and high operating amperage.

12.1.4 Encoders

The encoders are sensors that increment as rotation in the system's wheels occur. This information is used to calculate the wheels position and velocity of the SmartWalker. This is vital to PID control of the system's velocity. The encoders will communication a PWM signal to the Arduino controller of its respective drive module.

12.1.5 Ultrasonic Sensors

The ultrasonic sensors send high frequency pulses out into the environment and uses the sound reflected back into it in order to determine the proximity of objects in its environment. The system will have four of these sensors mounted in order to collect proximity data from four different directions (not including data from the Kinect). This object detection will aid the system in object avoidance during its navigation. The ultrasonic sensors will communicate directly with an Arduino controller, which then will relay the data to the Raspberry Pi via serial communication.

12.1.6 Potentiometers

The potentiometers are used to provide closed loop feedback to the Arduinos to engage and disengage the brake motors. The potentiometers will communicate with its respective Arduino over a PWM signal. There will be one sensor in each wheel module for a total of two potentiometers.

12.2 High Voltage

The high voltage circuit is utilized to power the entire system. A 12V battery is connected to a power distribution board which powers the drive motors through the motor controller boards, it also powers all the microprocessor boards. All boards require sufficient power to operate correctly even if the communication between them use low voltage circuitry. ***Figure 10 - High Voltage Circuit*** depicts all the hardware units with descriptions and specifications of each.

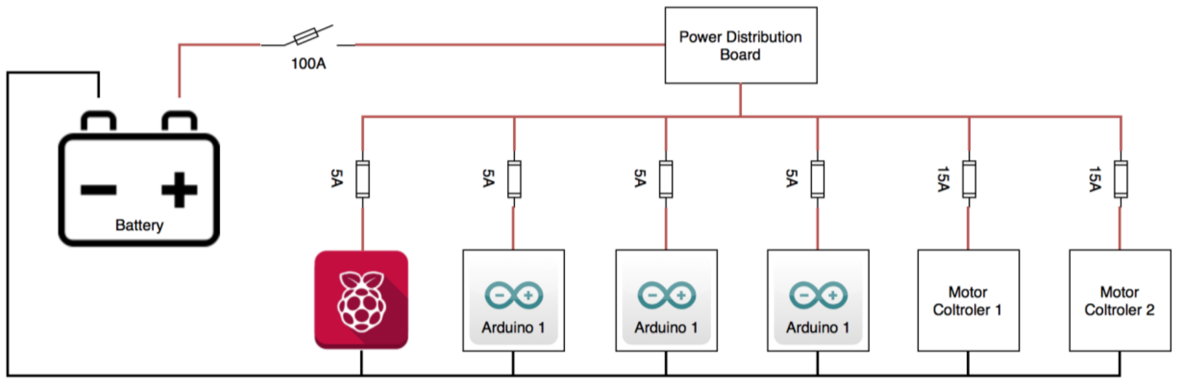

Figure 10 - High Voltage Circuit

12.2.1 Raspberry Pi 3B

The Raspberry Pi is the main controller for the system. It will house the main code for the ROS libraries that we have chosen to control the SmartWalker with. The Raspberry Pi 3 was specifically chosen due to its wireless LAN and Bluetooth connectivity. The Pi is small, efficient, and powerful enough to run the main functions we require for the SmartWalker. These functions include navigating of the SmartWalker, the HMI, and the computation of the data received from the Kinect. They are further discussed in section 5 Software Component.

12.2.2 Motor Controller Board

The motor controller boards are powered from the onboard battery. Once a signal is received, the board uses its onboard circuitry to drive the motors. Since we have two different types of motors we will have two different types of motor controller boards.

12.2.3 Arduino Micro

There are three arduinos being used as controllers in the system. Two of them are for controlling the drive modules, one for the left side and one for the right. These two are responsible for the real-time PID control for the speed of the motors. The third is collecting environment proximity data from the ultrasonic sensors and the desired drive/brake input from the potentiometers on the SmartWalker's handles. This data is handed off to the Raspberry pi for use in object avoidance and navigation control. The Arduino Micro's were chosen to offload some of the processing power from the Raspberry Pi, they are small, and have the capabilities of real-time control.

12.2.4 Battery

A 12V battery is used to power all components of the system, an appropriate 18Ah battery was chosen after current draw calculations were done for all units of the system.

12.2.5 Power Distribution Board

The power distribution board takes the 12V Battery as an input and then powers all the high voltage components of the system. Appropriate sized fuses are connected to the power distribution board to prevent damage to equipment. This allows the system to require only one battery to power both the low voltage and high voltage circuits since the low voltage circuit components will be powered from the low voltage power regulators built into the Raspberry Pi and Arduinos. Having one battery per system allows for easy maintenance and recharge.

13 Software Components

The software architecture of the SmartWalker project is divided into several modules or nodes which are used within each software component of the SmartWalker. The system comprises of ROS libraries, Arduino code, a simple phone application, and the Estimote Robotics Indoor SDK. An overview of the software architecture can be seen in the figure below.

ROS which stand for robot operating system, has all the functionality of a operating system but is not a stand-alone OS. Rather, it is composed of a group of software libraries and tools that aid in the creation of robotics. The Raspberry Pi houses a 32 GB SD card running UbuntuMATE and on top of that we have ROS, Kinect distribution.

Each subsection is a module of the software component of the system. The following is a guideline for the breakdown of each module.

1. Purpose of the module
2. Scope
3. Module Guideline – usually in the form of a diagram
4. Module Interface Specification (MIS) - definition of behaviour of each “method in terms of inputs and outputs - as black box as feasible”
    - Includes: responsibilities, inputs, outputs, and secrets
5. Module Internal Design (MID) - enough detail to make the coding close to trivial
6. Scheduling or Timing Constraints
7. Data Dictionary – definition of specific terms used in module ex. ROS messages

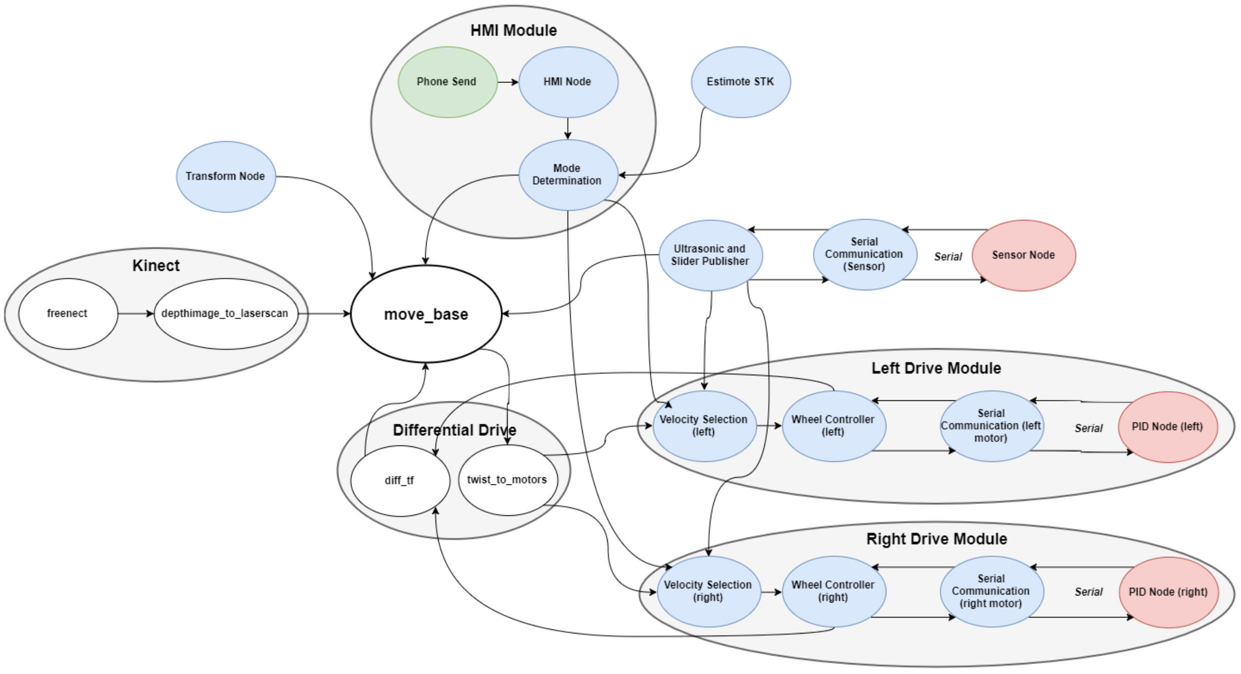

Figure 11 - Software System Map

13.1 Ultrasonic and Sensor Nodes

13.1.1 Purpose

To publish the location of surfaces sensed by the ultrasonic sensors and to publish the position of the sliders.

13.1.2 Scope

The *Ultrasonic and Slider Publisher* will separate the information from the *serial communication node* back into separate values for the ultrasound magnitudes and the slider values. The slider values will be published to the *drive module*. The ultrasound magnitudes will be converted into points (where are translations in meters) in space with a single coordinate frame. These points will then be published as a single point cloud to *move_base*.

The *serial communication node* will send a message on initialization to tell the *sensor node* to begin transmitting information. The *serial communication node* in ROS will read in the data transmitted by the *sensor node* and publish it to the *Ultrasonic and Slider Publisher* node.

There *sensor node* on the Arduino will excite the ultrasound sensors such that the distance to the nearest surface along the sensing axis of each sensor in centimeters can be read. It will also read values from the slider sensors that correspond to the position of each slider. These values will periodically be transmitted over serial communication to the Raspberry Pi.

13.1.3 Module Guideline

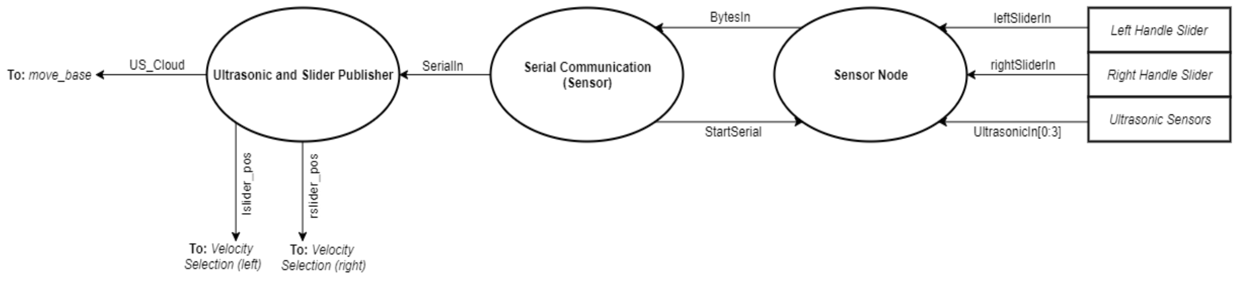

Figure 12 - Ultrasonic and Sensors Node Guideline

13.1.4 Module Interface Specification

Ultrasonic and Slider Publisher

Inputs

Name	**Type**	**Description**
SerialIn	std_msgs/String	Contains integers values for the ultrasonic sensor magnitudes and slider position encoded as characters in a string

Outputs

Name	**Type**	**Description**
US_Cloud	sensor_msgs/PointCloud	The magnitude reading from each ultrasonic sensor represented as 3D points in space within the Ultrasonic_frame coordinate frame
lslider_pos	std_msgs/UInt16	The left slider position represented as an integer
rslider_pos	std_msgs/UInt16	The right slider position represented as an integer

Serial communication (sensor)

Inputs

Name	Type	Description
BytesIn	Byte Array	The bytes read from the serial communication messages representing slider and ultrasonic data.

Outputs

Name	Type	Description
SerialIn	std_msgs/String	The converted string of bytes from serial to be sent out to the ultrasonic and slider publisher.
StartSerial	Byte	The start serial communication signal sent to the sensor node on the Arduino..

Sensor node

Inputs

Name	Type	Description
EchoIn[0:3]	Float32	The duration (in microseconds) it took for the trigger pin to send a signal and the echo pin to receive the feedback. 4 values for four ultrasonic sensors.
leftSliderIn	Float32	The voltage across left linear potentiometer.
rightSliderIn	Float32	The voltage across right linear potentiometer.
StartSerial	Byte	Byte representing a request to begin send serial messages.

Outputs

Name	Type	Description
BytesIn	Byte Array	The array of bytes containing slider and ultrasonic sensor data.

Secrets

Name	Type	Description
distance_cm[0:3]	UInt16	The proximity distance to objects in line of each ultrasonic sensor (in cm).
leftSliderOut	UInt16	The mapped left input slider value to integers between 0 and 1024.
rightSliderOut	UInt16	The mapped rightinput slider value to integers between 0 and 1024.

13.1.5 Module Internal Design

Ultrasonic and Slider Publisher

Decoding

Whenever a new SerialIn is published, its characters must be decoded into integers according to the following table:

Character index	Description of value
0	lslider_pos LSB
1	lslider_pos MSB
2	rslider_pos LSB
3	rslider_pos MSB
4	Ultrasonic sensor 1 magnitude LSB
5	Ultrasonic sensor 1 magnitude MSB
6	Ultrasonic sensor 2 magnitude LSB
7	Ultrasonic sensor 2 magnitude MSB
8	Ultrasonic sensor 3 magnitude LSB
9	Ultrasonic sensor 3 magnitude MSB
10	Ultrasonic sensor 4 magnitude LSB
11	Ultrasonic sensor 4 magnitude MSB

Table 11 - Ultrasonic and Slider Publisher Decoding

This decoding should be done according to the equation:

$$(char_{MSB} << 8) \quad || \quad char_{LSB}$$

Equation 2 - Ultrasonic and Slider Publisher Decoding

Conversion of ultrasonic measurements to points

The measurements from the ultrasonic sensors represent the distance from the ultrasonic sensor to the closest surface along its sensing axis in centimeters. The goal is to convert these magnitudes into 3-D points in a single coordinate frame, where the axis translations are in meters. To accomplish this a pose (x, y, z, theta) in space is chosen as the base frame of the ultrasonic sensors. By considering the magnitude measurements to be along the x-axis of the corresponding ultrasonic sensor, an array easily be formed that defines the difference in pose between that sensor and the ultrasonic coordinate frame (a transform) for each ultrasonic sensor. These transforms can then be used to change the magnitude measurements into points in space.

Define tf as the transform for an ultrasonic sensor, which stores a pose difference in terms of the x, y, z, and theta axis.

Point coordinate	Equation
X coordinate	$\mathrm{tf}_x + \frac{magnitude}{100} \times \cos(\mathrm{tf}_{theta})$
Y coordinate	$\mathrm{tf}_y + \frac{magnitude}{100} \times \sin(\mathrm{tf}_{theta})$
Z coordinate	tf_z

Table 12 - Conversion of Ultrasonic Measurement to Points

The 3-D points for each ultrasonic sensor should be combined into a point cloud message and published along with the decoded slider positions.

Serial communication (sensor)

<u>Publishing slider and ultrasonic bytes array:</u>

1. Initialize temporary string variable, include serial communication library.
2. Read in bytes array from serial, convert and assign to a string message and then publish data over a ROS topic.

<u>Subscribing to ROS topic which sends the SendSerial message:</u>

1. Subscribe to ROS topic message string, convert to bytes and write bytes over serial communication.

Sensor node

<u>Calculating magnitude from ultrasonic sensors:</u>

1. Send a digital high on each trigger pin, wait 10 microseconds, then send a digital low on each trigger pin.
2. Read the value from echo pin (time) and convert to a distance magnitude.

Distance magnitudes from each ultrasonic sensor are calculated using the following formula:

$$distance\_cm[i] = EchoIn[i] * 0.034/2$$

Equation 3 - Calculating Magnitude from Ultrasonic Sensors

Where *i* is the ultrasonic sensor number.

<u>Determining value from each slider:</u>

1. Read in the analog voltage signal from left and right linear potentiometer.
2. Map value from potentiometer minimum and maximum to integer value between 0 and 1024.

<u>Converting secrets to output bytes array:</u>

1. Convert the left slider output value to bytes, send the LSB, then the next least significant byte.
2. Convert the right slider output value to bytes, send the LSB, then the next least significant byte.
3. Convert each distance output from each ultrasonic sensor to bytes, send the LSB, then the next least significant byte.

13.1.6 Scheduling

Timing Requirements

1. The slider values are to be published at

13.1.7 Data Dictionary

The following table defines the ROS messages that are used in this node. Each message class may use primitive types, but they will not be defined.

ROS Message Type	Format	Definition
std_msgs/String	string data	Class in ROS that holds data of type string
std_msgs/UInt16	uint16 data	Class in ROS that holds data of type uint16
sensor_msgs/PointCloud	Header header geometry_msgs/Point32[] points	An array of 3D points where each point is a Point32 type based on the frame defined in header.
geometry_msgs/Point32	float32 x float32 y float32 z	Position of a point in free space.

Table 13 - Ultrasonic and Sensor Node Data Dictionary

13.2 Drive Module(s)

The drive module will be described in general rather than one module per wheel because they can be controlled separately.

13.2.1 Purpose

To move each wheel of the walker at the commanded velocity if the walker is in autonomous or assisted mode. To allow the walker to move freely if the walker is in free-roll mode.

13.2.2 Scope

The *velocity selection* node differentiates between the autonomous and manual inputs for wheel velocity which correspond to a specific state of operation the SmartWalker would be in. It receives velocity commands, and values from the handle sliders, electronic brake button and state of operation topics. This node sends the selected velocity command as a float to the appropriate wheel controller node.

The *wheel controller* node takes in the velocity command for a wheel and convert it into a format suitable for serial communication to the Arduino. It also builds encoder information about the wheel from the data received from the arduino and then publishes it.

The *serial communication* node for the drive modules is to read and write serial messages between the raspberry pi controller and the arduino controllers. The node reads bytes from serial messages and converts it to a string to be sent as a topic to other nodes. It also receives a topic of strings which is converted to bytes to be written over serial communication to another node.

The *PID* node takes in the velocity command over serial communication and encoder information for wheel position in order to determine a PID controller output to be sent as a pulse width modulation signal to the motor control board. This node uses the velocity command from serial as a PID setpoint and calculates the current wheel velocity from the encoder positions to be used as the main PID input.

13.2.3 Module Guideline

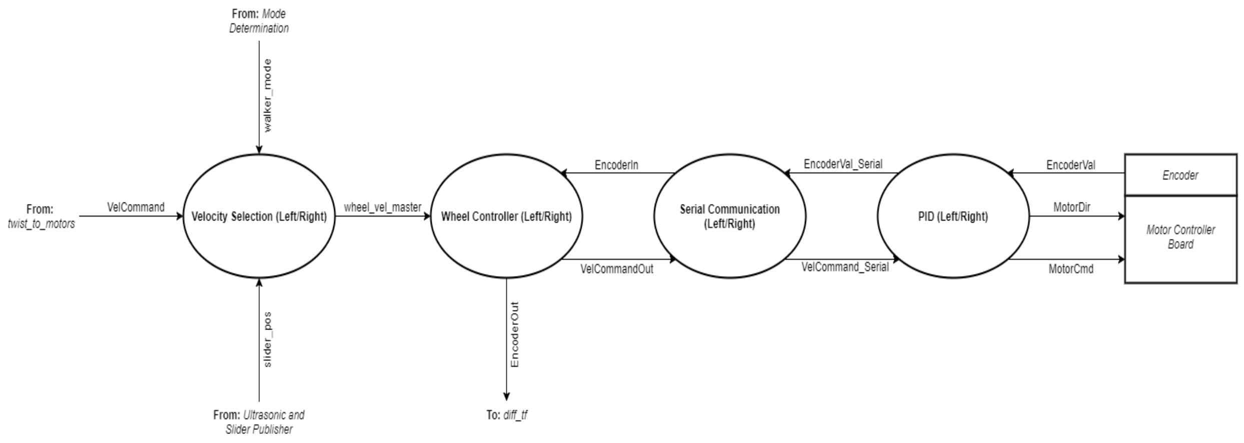

Figure 13 - Drive Module Guideline

13.2.4 Module Interface Specification

Velocity Selection

Inputs

Name	Type	Description
VelCommand	std_msgs/Float32	The velocity command for the wheel from the twist_to_motors node.
slider_pos	std_msgs/UInt16	The slider position for the corresponding wheel
walker_mode	std_msg/UInt16	Describes whether the walker is in autonomous or manual mode 0 - manual 1 - autonomous

Outputs

Name	Type	Description
wheel_vel_master	std_msgs/Float32	The velocity to be actuated by the wheel

Wheel Controller

Inputs

Name	Type	Description
wheel_vel_master	std_msgs/Float32	The velocity command for the wheel from the velocity selection node.
EncoderIn	std_msgs/String	The encoder value in the form of a string, which was formatted from a byte array representing an integer value.

Outputs

Name	Type	Description
VelCommandOut	std_msgs/String	The converted float velocity command into a bytes array formatted into a string.
EncoderOut	std_msgs/Int16	The encoder position converted from the formatted string of bytes into an integer value.

Serial Communication

Inputs

Name	Type	Description
BytesIn	Byte Array	The array of bytes read from the serial communication messages representing the encoder data.
VelCommandOut	std_msgs/String	The converted float velocity command into a bytes array formatted into a string.

Outputs

Name	Type	Description
SerialIn	std_msgs/String	The converted string of bytes from serial to be sent out to the ultrasonic and slider publisher.
StartSerial	Byte	The start serial communication signal sent to the sensor node on the Arduino.

PID

Inputs

Name	Type	Description
VelCommand_Serial	Byte Array	The velocity command read from a serial message to be converted to float value.
EncoderVal	Int16	The encoder value (wheel position) read directly as an analogue input.

Outputs

Name	Type	Description
MotorCmd	PWM Duty Cycle	The signal sent to the motor controller board to actuate the motor actuators on the wheel.
MotorDir	UInt16	The signal sent to the motor controller board which signifies the direction the motors actuate.
EncoderVal_Serial	Byte Array	The encoder value sent over serial communication to the Raspberry Pi.

13.2.5 Module Internal Design

Velocity Selection

1. If the walker is in autonomous mode (walker_mode == 3), publish the VelCommand input as wheel_vel_master.
2. If the walker is in manual mode (walker_mode == 1 || walker_mode == 2), create a velocity based on the slider position value according to the following table. Publish this velocity as wheel_vel_master.

Value of slider position	**Corresponding value for wheel_vel_master**
sliderVal <= 407	$max\_velocity \times (\frac{sliderVal}{407} - 1)$
sliderVal >= 408 && sliderVal <= 615	0
sliderVal > 615	$max\_velocity \times (\frac{sliderVal - 607}{407})$

Table 14 - Velocity Selection Calculation

Wheel Controller

1. If the walker is in autonomous mode, publish the wheel_vtarget from the twist_to_motors node.
2. If the walker is in manual mode, create a velocity command in m/s based on the slider position value based on the following table. Publish the velocity command.

Serial Communication

Publishing Encoder bytes array:

1. Initialize temporary string variable, include serial communication library.
2. Read in bytes array from serial, convert and assign to a string message and then publish data over a ROS topic.

Subscribing to ROS topic which sends the VelCommnadOut message:

1. Subscribe to ROS topic message string, convert to bytes and write bytes over serial communication.

PID Node

1. Receive Inputs.
2. Convert VelCommand_Serial into a 32 bit float value.
3. Calculate current wheel velocity by taking the EncoderVal at two time instances to calculate the wheel's angular velocity, then multiply the angular velocity by the radius of the wheel.
4. With the set point and input (VelCommand_serial and EncoderVal respectively), compute output PID signal.
5. Add PID output to the previous MotorCmd sent for new MotorCmd value. Take sign of this value as MotorDir value.
6. Send outputs.

13.2.6 Scheduling

Timing Requirements

1. Operate at a speed such that wheel encoder readings (EncoderVal) are constantly being read at least every 30 ms.
2. Operate at a speed such that wheel motor commands (MotorCmd, MotorDir) are constantly being written at least every 30 ms.
3. PID which controls MotorCmd output signal has a maximum settling time 1s.

13.2.7 Data Dictionary

The following table defines the ROS messages that are used in this node. Each message class may use primitive types, but they will not be defined.

ROS Message Type	**Format**	**Definition**
std_msgs/String	string data	Class in ROS that holds data of type string.
std_msgs/UInt16	uint16 data	Class in ROS that holds data of type uint16.
std_msgs/Int16	int16 data	Class in ROS that holds data of type int16.
std_msgs/Float32	Float32 data	Class in ROS that holds data of type Float32.
Variable	**Format**	**Definition**
walker_mode	UInt16	Init == 0, Freeroll = 1, Assisted = 2, Autonomous == 3.

Table 15 - Drive Module Data Dictionary

13.3 Differential Drive Module

13.3.1 Purpose

To generate left and right wheel velocity targets based on the overall velocity target given by move_base. To generate the odometry information required by move_base from the encoder values of each wheel.

13.3.2 Scope

Differential Drive is composed of two nodes, *diff_tf* and *twist_to_motors*, from the 'differential_drive' open source ROS package.

diff_tf is to read from the left and right encoder topics published by the left and right drive modules. Using these values, the current system time, and previous state information, an odometry signal will be created and published for use by move_base. Additionally, a transform from the base_link frame to the odometry frame will be published for use by move_base.

twist_to_motors is to read the twist command from move_base and convert it into target velocities for the left and right wheels for publication to the corresponding drive modules.

13.3.3 Module Guideline

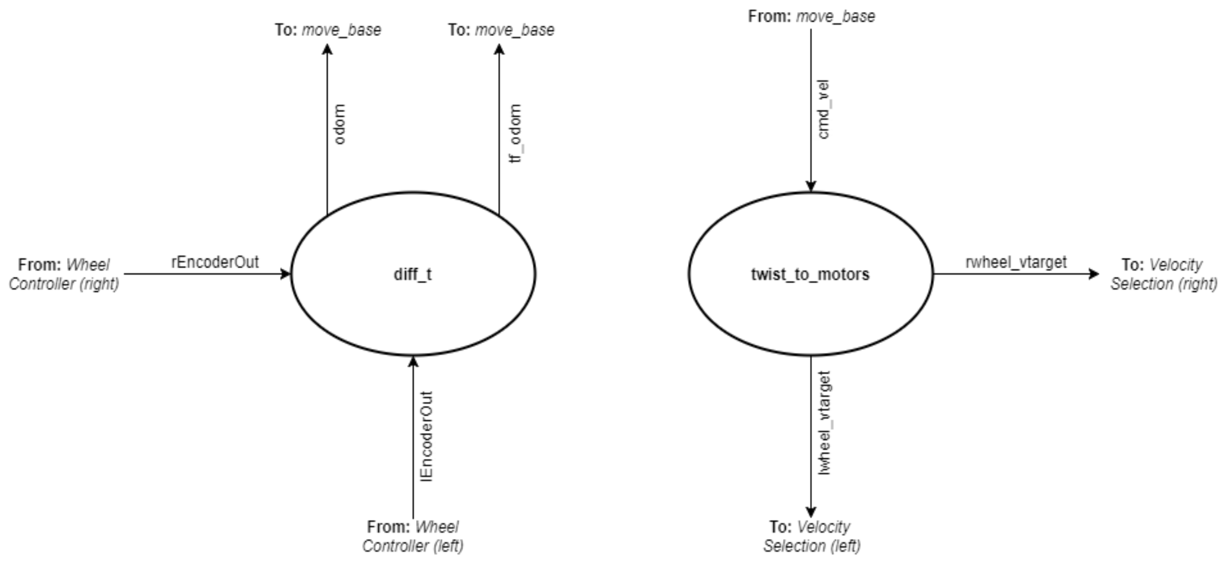

Figure 14 - Differential Drive Module Guideline

13.3.4 Module Interface Specification

Twist_to_motors

Inputs

Name	Type	Description
cmd_vel	geometry_msgs/Twist	The linear and angular velocity commands from move_base.

Outputs

Name	Type	Description
rwheel_vtarget	std_msgs/Float32	The target velocity for the right wheel in meters per second.
lwheel_vtarget	std_msgs/Float32	The target velocity for the left wheel in meters per second.

diff_tf

Inputs

Name	Type	Description
lEncoderOut	std_msgs/Int16	The encoder value for the left wheel.
rEncoderOut	std_msgs/Int16	The encover value for the right wheel.

Outputs

Name	Type	Description
tf_odom	tfMessage	The transform from the base_link frame and the odometry frame.
odom	nav_msgs/Odometry	The odometry of the robot.

13.3.5 Module Internal Design

No module internal design is provided because *differential drive* is composed of existing software.

13.3.6 Scheduling

Timing Requirements

1. Wheels must respond together for optimal driving operation

13.3.7 Data Dictionary

The following table defines the ROS messages that are used in this node. Each message class may use primitive types, but they will not be defined.

Message Type	Format	Definition
tf/tfMessage	geometry_msgs/TransformStamped[] transforms	See geometry_msgs/TransformStamped
geometry_msgs/TransformStamped	Header header string child_frame_id Transform transform	This expresses a transform from coordinate frame header.frame_id to the coordinate frame child_frame_id with header being used to communicate timestamped data
geometry_msgs/transform	Vector3 translation Quaternion rotation	This message type represents the transform from one point in space to another. Used in the overall tfMessage class.
nav-msgs/Odometry	Header header string child_frame_id geometry_msgs/PoseWithCovariance pose geometry_msgs/TwistWithCovariance twist	This is an estimate of the position and velocity in the environment. The pose should be specified by the coordinate frame in the header and the twist by the child frame. Pose being the position and orientation. Twist being the velocity broken down into linear and angular parts.
geometry_msgs/Twist	Vector3 linear Vector3 angular	Linear and angular parts used to describe the velocity.
std_msgs/Int16	int16 data	Class in ROS that holds data of type int16
std_msgs/Float32	Float32 data	Class in ROS that holds data of type Float32

Table 16 - Differential Drive Data Dictionary

13.4 Kinect Module

13.4.1 Purpose

To produce a laserscan topic from the depth-image data of a Microsoft Kinect for *move_base* to use for perception purposes during autonomous navigation. This is necessary because the basic requirement of move_base is a planar laser scanner, but such scanners that fit the project budget are significantly less accurate than the data that the Kinect can provide.

13.4.2 Scope

A Microsoft Kinect will be used to get depth-image data that will be converted into laserscan for use by *move_base*. The *freenect stack*, an open source ROS driver for the Microsoft Kinect, is utilized to generate ROS topics for the depth camera info and depth images. The rate at which depth-images are published will be controllable by setting a parameter.

depthimage_to_laserscan, an open source ROS package, is used to convert the depth-image from the Kinect into a laserscan message suitable for use with move_base.

13.4.3 Module Guideline

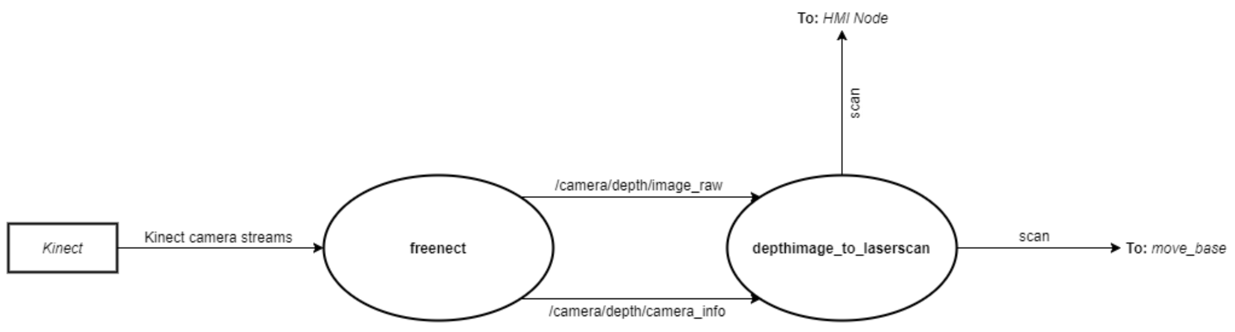

Figure 15 - Kinect Module Guideline

13.4.4 Module Interface Specification

freenect stack

Inputs

Name	Type	Description
Kinect camera streams	N/A	Depth and colour image data from the kinect.

Outputs

Name	Type	Description
/camera/depth/image_raw	sensor_msgs/Image	A cloud of 3-D points that represent surface locations in the environment within the Kinect's field of view.
/camera/depth/camera_info	sensor_msgs/CameraInfo	Information about the camera for the associated image.

depthimage_to_laserscan

Inputs

Name	Type	Description
/camera/depth/image_raw	sensor_msgs/Image	A cloud of 3-D points that represent surface locations in the environment within the Kinect's field of view.
/camera/depth/camera_info	sensor_msgs/CameraInfo	Information about the camera for the associated image.

Outputs

Name	Type	Description
scan	sensor_msgs/LaserScan	The laserscan representation of the depth image data from the kinect.

13.4.5 Module Internal Design

freenect_stack parameters

Name	Type	Description
data_skip	Integer	Sets how many images should be not published for everyone that is published. Effectively allows you to lower the 'framerate' of the data.

An MID is otherwise not provided because the freenect stack is existing software.

13.4.6 Scheduling

1. The Kinect module will publish a laserscan message to move_base at least 5 times a second

a. The rate will be increased if the Raspberry Pi hardware is capable of it. The rate was set based on computational limits of the Pi in regards to working with a Kinect that were found online.
b. The rate that image data comes from the freenect module can be controlled via the data_skip parameter

13.4.7 Data Dictionary

The following table defines the ROS messages that are used in this node. Each message class may use primitive types, but they will not be defined.

ROS Message Type	**Format**	**Definition**
sensor_msgs/Image	Header header uint32 height uint32 width	The header in this message holds various types of information. The timestamp should be acquisition time of image, the frame_id should be optical frame of camera, the origin of frame should be optical center of camera, the +x should point to the right in the image, the +y should point down in the image, the +z should point into to plane of the image. Height being the number of rows and width being the number of columns in the image.
sensor_msgs/CameraInfo	Header header uint32 height uint32 width string distortion_model float64[] D float64[9] K float64[9] R float64[12] P uint32 binning_x uint32 binning_y RegionOfInterest roi	This message defines meta information for a camera. The header will give the same information as the sensor_msgs/image, if there is a conflict, the information will be undefined. Height and width represent the resolution of the camera. Float64 are different calibration paramenters of the camera with which the image was captured, represented in 3x3 matrix form. Binning describes the camera setting process of combining rectangular neighborhoods of pixels into larger “super-pixels” usually reducing the resolution. The RegionOfInterest is described below.
sensor_msgs/RegionOfInterest	uint32 x_offset uint32 y_offset uint32 height uint32 width	This message is used to specify a region of interest within an image. x_offset and y_offset describe the left and top most pixel of the image. The height and width being the number of

	bool do_rectify	rows and columns and do_rectify is true when the whole image is selected as the ROI.

13.5 Transform Module

13.5.1 Purpose

To publish transforms from the 'base_link' frame to the coordinate frame of the kinect and ultrasound sensors.

13.5.2 Scope

The *Transform Node* will publish a transform from 'base_link' to 'Kinect_frame' for the Kinect and another from 'base_link' to 'Ultrasonic_frame' for the ultrasonic sensors. Rotations will be described in terms of rotations about the x, y, and z axis. The 'Ultrasonic_frame' used in the transform for the ultrasonic sensors should be the same as the base frame used for the ultrasonic sensors in the *Ultrasonic and Slider Publisher*

13.5.3 Module Guide

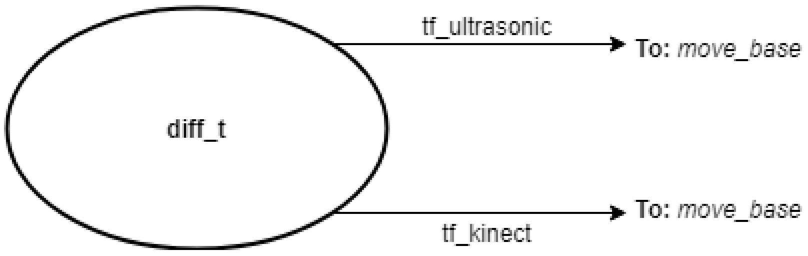

Figure 16 - Transform Node Module Guideline

13.5.4 Module Interface Specification

<u>Outputs</u>

Name	Type	Description
tf_ultrasonic	tf/tfMessage	The transform from the base frame of the walker to the frame of the ultrasonic sensors
tf_kinect	tf/tfMessage	The transform from the base frame of the walker to the frame of the Kinect

13.5.5 Module Internal Design

The software is composed of two individual transform broadcasters. The broadcasters are instantiated with four parameters.

Parameter number	Name	Type	Description
1	Quaternion	tf::Quaternion	Describes the rotation of the child frame about the parent frame's x, y, and z axis
2	Vector	tf::Vector3	Describes the translation of the child frame about the parent frame's x, y, and z axis
3	Parent frame	String	Name of the parent frame
4	Child frame	String	Name of the child frame

The parameters for each transform should be set to the values given in the following table.

Note: The values for the quaternion and vector will be filled in once the sensors are in their final locations.

Transform name	Quaternion in radians about (x,y,z)	Vector in meters about (x,y,z)	Parent frame	Child frame
tf_ultrasonic	N/A	N/A	"base_link"	"Ultrasonic_frame"
tf_kinect	N/A	N/A	"base_link"	"Kinect_frame"

13.5.6 Scheduling

13.5.7 Data Dictionary

The following table defines the ROS messages that are used in this node. Each message class may use primitive types, but they will not be defined.

ROS Message Type	**Format**	**Definition**
tf/tfMessage	geometry_msgs/TransformStamped[] transforms	See geometry_msgs/TransformStamped
geometry_msgs/TransformStamped	Header header string child_frame_id	This expresses a transform from coordinate frame header.frame_id to the coordinate

	Transform transform	frame child_frame_id with header being used to communicate timestamped data
geometry_msgs/transform	Vector3 translation Quaternion rotation	This message type represents the transform from one point in space to another. Used in the overall tfMessage class.
geometry_msgs/Point32	float32 x float32 y float32 z	Position of a point in free space.

13.6 HMI Module

13.6.1 Purpose

To receive human-machine interface (HMI) inputs from the walker interface or from the mobile smartphone interface so that user can select a specific target for autonomous mode and select between assisted mode or free-role mode within manual operation.

13.6.2 Scope

The *phone sending* node (Android phone application) sends a command to the Raspberry Pi using Bluetooth communication. This node acts as a Bluetooth client searching for a connection with a server. Sends data to the Raspberry Pi depending on the command selected by the user via Android phone application.

The *HMI node* will display system information and allow the user to request braking commands through an LCD touchscreen. The graphical user interface on the touch screen will display battery level, determine and display safety status and navigation information. The node contains buttons that will allow the user to switch between free-rolling and braking modes. This node directly operates on the Raspberry Pi. This node also interprets the data sent from an Android phone using Bluetooth communication and determines which command should be executed. This node acts as a Bluetooth server awaiting connection from a client.

The *mode determination* node determines the state of operation of the walker system. By receiving inputs to determine the appropriate mode and mapping information from the estimote stk node, the publishes the mode of operation and sends the target location to move_base if the mode is in autonomous.

13.6.3 Module Guideline

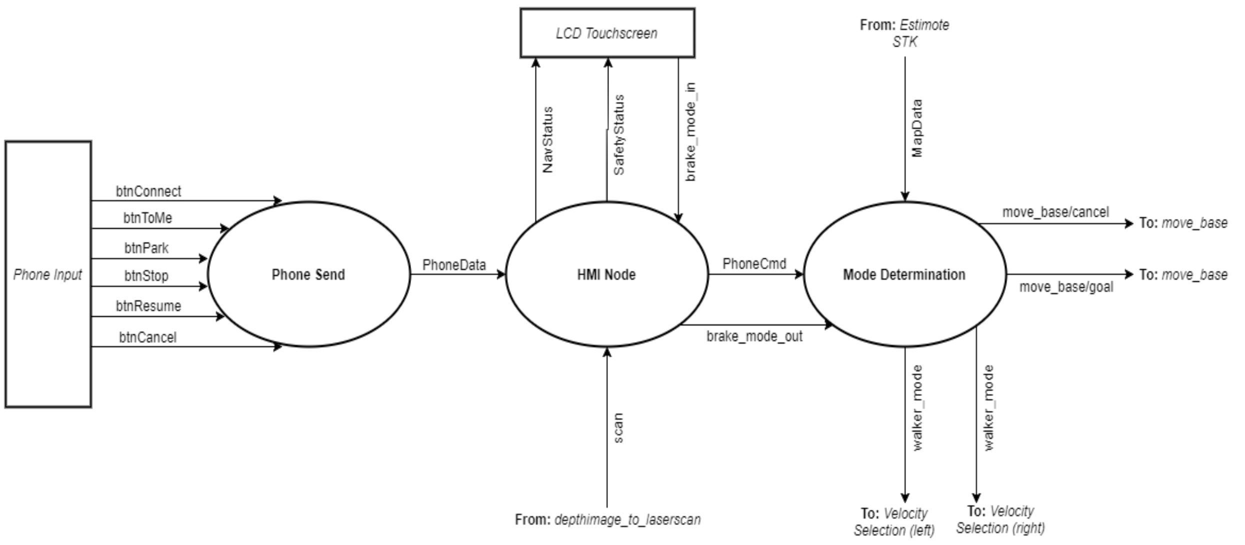

Figure 17 - HMI Module Guideline

13.6.4 Module Interface Specification

Phone Send

Inputs

Name	Type	Description
btnConnect	android.widget.Button	Android button widget, when clicked attempts to create a Bluetooth connection between the Android phone application and Raspberry pi
btnToMe	android.widget.Button	Android button widget, when clicked the application will send an integer value corresponding to a command to the Raspberry pi
btnPark	android.widget.Button	Android button widget, when clicked the application will send an integer value corresponding to a command to the Raspberry pi
btnStop	android.widget.Button	Android button widget, when clicked the application will send an integer value corresponding to a command to the Raspberry pi

btnResume	android.widget.Button	Android button widget, when clicked the application will send an integer value corresponding to a command to the Raspberry pi
btnCancel	android.widget.Button	Android button widget, when clicked the application will send an integer value corresponding to a command to the Raspberry pi

Outputs

Name	**Type**	**Description**
PhoneData	byte array	Data sent Raspberry Pi via Bluetooth

Secrets

Name	**Type**	**Description**
txtCmd	android.widget.TextView	Android textview widget, when a btn is clicked txtCmd will display which command is currently being sent.

HMI Node

Inputs

Name	**Type**	**Description**
brake_mode	int	integer value reading the state of the brake button
PhoneData	byte array	Data received from Android phone via Bluetooth
scan	sensor_msgs/LaserScan	The laserscan representation of the depthimage data from the kinect

Outputs

Name	Type	Description
PhoneCmd	Int32	Integer value corresponding to a command.

Mode Determination

Inputs

Name	Type	Description
PhoneCmd	Int32	Integer value corresponding to a command.
brake_mode_out	UInt16	Integer value reading state of brake button.
MapData	2D Float32 Array	Array of coordinates representing beacons on an automatically generated map.

Outputs

Name	Type	Description
goal	geometry_msgs/PoseStamped	The target location and target pose of the walker during autonomous navigation.
walker_mode	UInt16	The mode of operation of the SmartWalker, either autonomous, or manual (free-roll or assisted).

13.6.5 Module Internal Design

Phone Send

1. All buttons are "listening" for the user.
2. User clicks a button.
3. If button is not btnConnect nothing happens. If btnConnect is clicked Bluetooth connection thread will initiate.
4. The thread will check which devices are currently paired the Android phone running the application and obtain the paired device's UUID (Universally Unique Identifier). It will then open a bluetooth socket using the UUID and create a Bluetooth connection. If there are no devices paired nothing happens.

5. If the connection was successful and the user selects any button (excluding btnConnect) data associated with the command in the form of a byte array will be sent to the device currently sharing a Bluetooth connection with the Android phone. If btnConnect is clicked nothing will happen.

HMI Node

1. Allocate Bluetooth socket.
2. Listen for connection.
3. Accept connection.
4. Receive input data.
5. Convert data to integer.
6. Return integer value.
7. Receive safety and navigation data and display on touchscreen
8. Display image-based button on touchscreen
9. Track touchpresses on the button on the touchscreen
10. Releasing touchscreen button sends data to the Pi to change brake mode output to mode determination node

Mode Determination

1. When turned on walker_mode == 0 (Initialization).
2. After Initialization is complete the walker will go to its default, walker_mode == 1 (Manual mode with free rolling wheels).
3. If brake_mode_out == walker_mode == 1 (Manual mode with free rolling wheels).
4. If brake_mode_out == walker_mode == 2 (Manual mode with assisted movement).
5. If brake_mode_out == 0 and if there is a phone command, walker_mode == 3, read in MapData from Estimode SDK module and convert to ROS geomoetry_msg/PoseStamped goal message and publish.

13.6.6 Schedule

Timing Requirements

1. Operate at a speed such that Bluetooth connection is established between the Android phone and Raspberry Pi and txtCmd updated within 2 seconds of btnConnect being clicked.
2. Operate at a speed such that data (PhoneData) is sent to and received by the HMI node and txtCmd updated within 500 milliseconds of btnTome, btnPark, btnStop, btnResume or btnCancel being clicked.
3. Operate at a speed such that data (PhoneCmd) is passed onto mode determination node with 200 milliseconds of data (PhoneData) being received from Android phone.

13.6.7 Data Dictionary

Phone Send

Data Type	**Description**
android.widget.TextView[1]	Subclass of android.view.View which is a subclass of the superclass java.lang.Object. It is a user interface feature that displays text to the user.
android.widget.Button[2]	Subclass of android.view.TextView which is a subclass of android.view.View which is a subclass of the superclass java.lang.Object. It is a user interface feature that can be clicked to initiate/interrupt a task.

ROS Message Type	**Format**	**Definition**
geometry_msgs/PoseStamped	geometry_msgs/PostStamped target_pose geometry_msgs/PostStamped base_position	The target pose and position for *move_base* so that it knows where to navigate.
Name	**Data Type**	**Expected Values**
brake_mode, brake_mode_out	UInt16	Init = 0, Freeroll_pressed = 1, Assisted_pressed == 2.
walker_mode	UInt16	Init = 0, Freeroll = 1, Assisted = 2, Autonomous = 3.

13.7 Estimote Indoor SDK Module

13.7.1 Purpose

To provide the position and orientation of the walker and the user's phone.

13.7.2 Scope

After setting up the room for operation in the Estimote cloud that is used by our phone application, the SDK should allow the SmartWalker application to access the (x,y) coordinates and orientation of the walker and the users phone in a unified coordinate system. These coordinates will be retrieved from calls to the Estimote cloud and sent to the Walker over bluetooth.

13.7.3 Module Guide

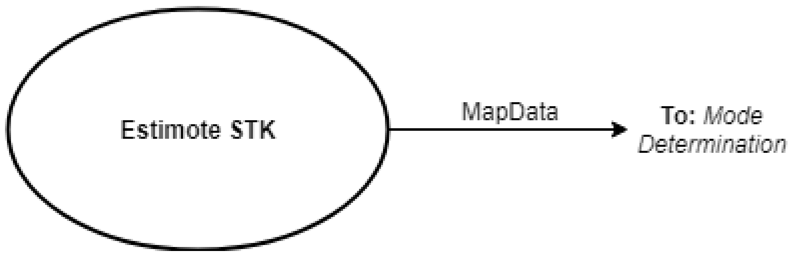

Figure 18 - Estimote Indoor SDK Module Guideline

13.7.4 Module Interface Specification

<u>Outputs</u>

Name	Type	Description
MapData	Float32 Array	Holds the (x,y) coordinates and orientation of the users phone and the walker.

13.7.5 Module Internal Design

An MID is not provided because the Estimote SDK is existing software

13.7.6 Scheduling

Timing Requirements

1. The SDK should produce the coordinate and orientation of the phone and the walker within 100 milliseconds
    a. Requirement comes from the system level requirement that the walker should begin navigation within half a second of the user generating a navigation command by interacting with the phone application

13.7.7 Data Dictionary

Name	**Data type**	**Data Format**	**Expected Values**
MapData	Float32 Array	● 2 rows (Phone, Walker) ● 3 Columns (x, y, theta) ● x and y are in meters ● theta is in radians	● X and Y will be below 20 ● Theta will be between 0 and 2pi

13.8 move_base Module

13.8.1 Purpose

To attempt to reach a given goal pose based on provided sensor and odometry information by producing a twist (linear and angular velocities) command for implementation by the walker.

13.8.2 Scope

move_base is an open source ROS package that is the core of the walker's navigation software.

Upon receiving a goal pose, *move_base* is to continuously utilize the laserscan data from the *Kinect* module, the US_cloud data from the *Ultrasonic and Slider Publisher* node, and odometry information from the *differential drive* package for avoiding objects and path planning to the goal pose. It attempts to implement its plan to reach the goal pose by continuously publishing a twist command for the other walker systems to implement.

Once the walker reaches the goal within a parameter-defined tolerance, *move_base* stops producing new twist commands until a new goal is received. Goals can be cancelled, which stops *move_base* from producing twist commands.

13.8.3 Module Guide

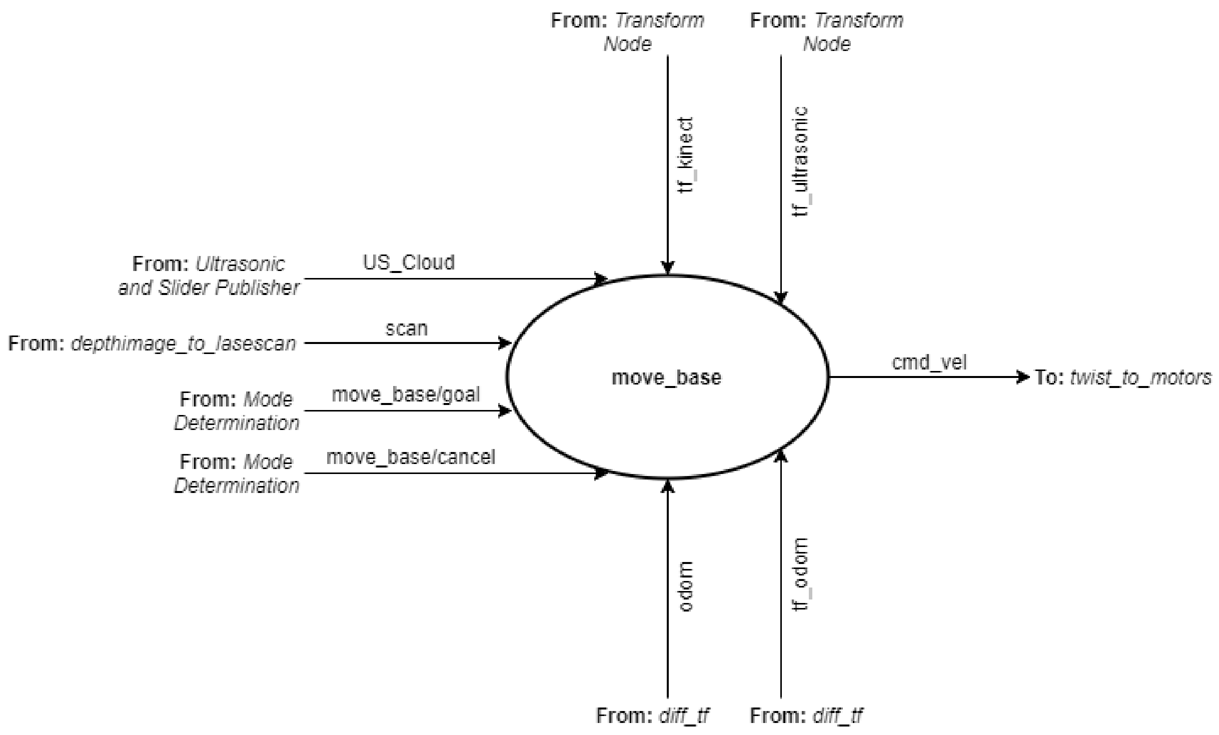

Figure 19 - move_base Module Guideline

13.8.4 Module Interface Specification

Inputs

Name	**Type**	**Description**
tf_ultrasonic	tf/tfMessage	The transform from the base frame of the walker to the frame of the ultrasonic sensors
tf_kinect	tf/tfMessage	The transform from the base frame of the walker to the frame of the kinect
tf_odom	tfMessage	The transform between the odometry frame and the robots mobile base frame
odom	nav_msgs/Odometry	The odometry of the robot
US_cloud	sensor_msgs/PointCloud	The magnitude reading from each ultrasonic sensor represented as 3D points in space within the Ultrasonic_frame coordinate frame
scan	sensor_msgs/LaserScan	The laserscan representation of the depthimage data from the kinect
move_base/goal	move_base_msgs/MoveBaseActionGoal	The 2-D location in space that the walker is supposed to navigate to
move_base/cancel	actionlib_msgs/GoalID	Tells move_base to stop trying to navigate towards the given goal

Outputs

Name	**Type**	**Description**
cmd_vel	geometry_msgs/Twist	The linear and angular velocity commands from move_base to be implemented by the walker

13.8.5 Module Internal Design

An MID is not provided because the move_base package is existing software.

13.8.6 Scheduling

Timing Requirements

1. Operate such that laser scan data from the *Kinect modul*e must be read at least every 30 milliseconds.
2. Operate such that US_cloud data from the *Ultrasonic and Slider Publisher* node must be read every 30 milliseconds.
3. Operate such that odometry information from the *differential drive* package must be read every 30 milliseconds.
4. Operate such that twist commands must be published to other walker systems every 30 milliseconds.

13.8.7 Data Dictionary

ROS Message Type	**Format**	**Definition**
tfMessage	geometry_msgs/TransformStamped[] transforms	See geometry_msgs/TransformStamped
geometry_msgs/TransformStamped	Header header string child_frame_id Transform transform	This expresses a transform from coordinate frame header.frame_id to the coordinate frame child_frame_id with header being used to communicate timestamped data
nav_msgs/Odometry	Header header string child_frame_id geometry_msgs/PoseWithCovariance pose geometry_msgs/TwistWithCovariance twist	This represents an estimate of a position and velocity in free space. The pose in this message should be specified in the coordinate frame given by header.frame_id. The twist in this message should be specified in the coordinate frame given by the child_frame_id Pose being the position and orientation. Twist being the velocity broken down into linear and angular parts.

sensor_msgs/PointCloud	Header header geometry_msgs/Point32[] points	An array of 3D points where each point is a Point32 type based on the frame defined in header.
geometry_msgs/Point32[]	float32 x float32 y float32 z	This contains the position of a point in free space(with 32 bits of precision).
sensor_msgs/LaserScan	Header header float32 angle_min float32 angle_max float32 angle_increment float32 time_increment float32 scan_time float32 range_min float32 range_max float32[] ranges float32[] intensities	Single scan from a planar laser range-finder. The timestamp in the header is the acquisition time of the first ray in the scan. In frame frame_id, angles are measured around the positive Z axis (counterclockwise, if Z is up) with zero angle being forward along the x axis Angle specifies the start and end angles of the scan as well as the angular distance between measurements in rads. Time increment specifies the time between measurements, and scan time specifies the time between scans. Range values specify the max and minimum range values, and range data is captured in array of 32-bit float
move_base_msgs/MoveBaseActionGoal	Header header actionlib_msgs/GoalID goal_id MoveBaseGoal goal	Send goal to navigation stack
actionlib_msgs/GoalID	time stamp string id	The stamp should store the time at which this goal was requested. It is used by an action server when it tries to preempt all goals that were requested before a certain time The id provides a way to associate feedback andresult message with specific

		goal requests. The id specified must be unique.
MoveBaseGoal goal	geometry_msgs/PoseStamped target_pose	Goal is requested using a Pose with reference coordinate frame and timestamp
geometry_msgs/Twist	Vector3 linear Vector3 angular	Linear and angular parts used to describe the velocity.

Table 17 - move_base Data Dictionary

Appendix I

Item #	Item	Specifications
1	Kinect	● Viewing angle: 43 vertical by 57 horizontal field of view ● Vertical tilt range: +/- 27 deg ● Frame rate (depth and colour stream): 30 FPS ● Audio format: 16-kHz, 24 bit mono pulse code modulation (PCM) ● Audio input characteristics: 4-microphone array with 24-bit ADC and Kinect-resident signal processing including acoustic echo cancellation and noise suppression ● Accelerometer characteristics: A 2G/4G/8G accelerometer configured for the 2G range, with a 1° accuracy upper limit.
2	12V Reversible Gear Motor	● Drive size: 5/32 in ● Voltage reading: 12V DC ● Rotation: reversible ● Weight: 0.9 lbs
3	65:1 25D Metal Gear Motor (6V 100RPM)	● Motor Rating @ 6V: ● 65:1 gear ratio ● 100RPM ● 50mA free run current ● 1.5A stall current ● 3.6kg*cm torque ● 4mm shaft diameter
4	Encoder	● Custom building an encoder, specifications to be determined
5	Estimote	● Battery life: 5 years ● Range: 200 m ● Thickness: 27 mm ● iBeacon or Eddystone packets: 8 simultaneously ● Additional Packets: connectivity, telemetry, user-defined ● Built-in sensors: motion, temperature, ambient light, pressure ● Estimote indoor location: automapping ● Additional tech: mesh networking, GPIO, RTC, RGB, LED, 1Mb EEPROM, programmable NFC
6	Ultrasonic Sensor	● Power Supply :+5V DC ● Quiescent Current : <2mA ● Working Current: 15mA

		● Effectual Angle: <15° ● Ranging Distance : 2cm – 400 cm/1" – 13ft ● Resolution : 0.3 cm ● Measuring Angle: 30 degree ● Trigger Input Pulse width: 10uS
7	Potentiometer	● Manufacturing part number: PTA2043-2015DPB103 ● Resistance: 10 kOhms ● Power: 0.1W ● tolerance +/- 20% ● Adjustment type: top, slider ● TravelrRange: 20 mm ● Mounting type: through hole
8	Raspberry Pi 3B	● Quad Core 1.2GHz Broadcom BCM2837 64bit CPU ● 1GB RAM ● BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board ● 40-pin extended GPIO ● 4 USB 2 ports ● 4 Pole stereo output and composite video port ● Full size HDMI ● CSI camera port for connecting a Raspberry Pi camera ● DSI display port for connecting a Raspberry Pi touchscreen display ● Micro SD port for loading your operating system and storing data ● Upgraded switched Micro USB power source up to 2.5A
9	Cytron 10A 5-25V Dual Channel DC Motor Driver	● Maximum continuous motor current: 10A ● Peak motor current: 30A ● Logic input (high level): 3 - 5.5V ● Logic input (low level): 0 - 0.5V ● Maximum PWM frequency: 20khz ● Motor driver with bi-directional control for 2 brushed DC motor ● Motor voltage range: 5V - 25V
10	Arduino Micro	● Processor: ATmega32U4 ● Operating/Input voltage 5V/7-12 V ● CPU Speed: 16 MHz ● Analog In/Out: 12/O ● Digital In/Out: 20/7 ● EEPROM [kB]: 1 ● SRAM [kB]: 2.5 ● Flash [kB]: 32

		• USB: micro • UART: 1
11	Power Supply	• Manufacturer: Interstate Batteries • Part #: SLA1116 • Weight: 12.5 pounds each battery • Size: 7.1" long x 3.0" wide x 6.58" tall • Volts: 12 • Amp hours: 18 • Life Span: ○ 3-5 years in float service application ○ 1-2 years in competition robotics at peak performance, but longer as a practice or demo battery • Type: Sealed Lead Acid • Case Material: ABS plastic • Charge Retention at 20 deg. C ○ 1 Month: 92% ○ 3 Months: 90% ○ 6 Months: 80% • Life Expectancy: ○ 100% Depth of Discharge: 200 Cycles ○ 80% Depth of Discharge: 225 Cycles ○ 50% Depth of Discharge: 500 Cycles
12	Reflective Optical Sensor 0.157" (4mm) PCB Mount	• Sensing Distance: 0.157" (4mm) • Sensing Method: Reflective • Voltage - Collector Emitter Breakdown (Max): 32V • Current - Collector (Ic) (Max): 50mA • Current - DC Forward (If) (Max): 50mA • Output Type: Phototransistor
13	LED Micro Blade Fuse Block	• Current rating: 25 A • Voltage rating: 32 V • 10 pc, waterproof

References

[1]"Raspberry Pi 3 Model B - Raspberry Pi", *Raspberry Pi*, 2018. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Accessed: 15- Jan- 2018].

[2]"Kinect for Windows Sensor Components and Specifications", *Msdn.microsoft.com*, 2018. [Online]. Available: https://msdn.microsoft.com/en-us/library/jj131033.aspx. [Accessed: 14- Jan- 2018].

[3]"Cytron 10A 5-25V Dual Channel DC Motor Driver", *Robotshop.com*, 2018. [Online]. Available: https://www.robotshop.com/en/cytron-10a-5-25v-dual-channel-dc-motor-driver.html#Specifications. [Accessed: 14- Jan- 2018].

[4]"12V Reversible Gear Motor | Princess Auto", *Princess Auto*, 2018. [Online]. Available: https://www.princessauto.com/en/detail/12v-reversible-gear-motor/A-p8365298e. [Accessed: 16- Jan- 2018].

[5]"65:1 25D Metal Gear Motor (6V 100RPM)", *Creatron Inc*, 2018. [Online]. Available: https://www.creatroninc.com/product/65-1-25d-metal-gear-motor-6v-100rpm/?search_query=motor&results=230. [Accessed: 12- Jan- 2018].

[6]"Set of 2 Batteries: Interstate Batteries (am-3062)", *www.AndyMark.com*, 2018. [Online]. Available: http://www.andymark.com/power-patrol-sla1116-12-volt-18ah-batteries-2-pack-p/am-3062.htm. [Accessed: 14- Jan- 2018].

[7]"PTA2043-2015DPB103 Bourns Inc. | Potentiometers, Variable Resistors | DigiKey", *Digikey.ca*, 2018. [Online]. Available: https://www.digikey.ca/products/en?keywords=PTA2043-2015DPB103-ND. [Accessed: 14- Jan- 2018].

[8]"Estimote", *Estimote.com*, 2018. [Online]. Available: https://estimote.com/products/. [Accessed: 16- Jan- 2018].

[9]V. Division, "TCRT1010 Vishay Semiconductor Opto Division | Sensors, Transducers | DigiKey", *Digikey.com*, 2018. [Online]. Available: https://www.digikey.com/product-detail/en/vishay-semiconductor-opto-division/TCRT1010/751-1032-ND/1681166. [Accessed: 12- Jan- 2018].

[10]"Arduino - Compare", *Arduino.cc*, 2018. [Online]. Available: https://www.arduino.cc/en/Products/Compare. [Accessed: 16- Jan- 2018].

[11]"25A LED Micro Blade Fuse Block | Princess Auto", *Princess Auto*, 2018. [Online]. Available: https://www.princessauto.com/en/detail/25a-led-micro-blade-fuse-block/A-p8479503e. [Accessed: 16- Jan- 2018].

[12]"Raspberry Pi Touch Display - Raspberry Pi", *Raspberry Pi*, 2018. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-touch-display/. [Accessed: 16- Jan- 2018].

[13]"Complete Guide for Ultrasonic Sensor HC - SR04", *Random Nerd Tutorials*, 2018. [Online]. Available: https://randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04/. [Accessed: 16- Jan- 2018].