Software Design Document

for

AARDVARK

Version 1.0 approved

Prepared by Hannah DeFazio, Nikolas Lamb, Ben Myrick, Jenna Ryan

Clarkson University

December 4, 2018

Contents

Revision History

Name	Date	Reason For Changes	Version
Hannah DeFazio, Nikolas Lamb, Ben Myrick, Jenna Ryan	10/23/18	Initial release	1.0

1 Introduction

1.1 Purpose

The purpose of this document is to describe the architecture and design principles for *AARDVARK*. Included will be a system overview, design of data, external interface design component design, and the actual human interface design.

The intended audience is developers and project leads. This document is intended to be further expanded upon as needed for any new features, or necessary edits to existing designs.

1.2 Scope

The objective of this document is to describe the design of every necessary feature and provide a road map for implementation purposes. Team leads shall find it straightforward to delegate tasks and developers will find implementation easy based on this document. Additionally, should further features be proposed, this document also describes the framework said feature should follow.

1.3 Overview

This document is layered in a systematic way starting with a high-level system overview. This overview describes how the system behaves and interacts internally. Next is the external interface design. This section describes the relationships between varying components and how they interact. Next is the data design section that describes how data is stored, saved, and shared. Next is the component design which goes into detail about each component and what should be done to implement each feature. Finally there is the human interface design, describing the design of the GUI.

2 System Overview

Our system will be communicating across different components in order to create a larger system that interacts together to create the overall intelligence of the robot. The different components handle Kinect data analysis, object avoidance, system login, and robot movement. The Kinect data analysis returns a 1 or a 0 based on if the user needs assistance. This gets delivered to the robot movement. The robot movement also takes into account the object avoidance in order to determine which direction to send the robot. The system login handles the user's credentials in order to protect the information used by AARDVARK.

3 System Architecture

Robot features are divided into three main operating states that are either reactive to user condition or proactive to maintain operation. Each operating state performs a vital function. The first state is default, that is other states will be enacted from this state and after they are completed the robot will return to the default state. Additional non-essential states may be added at a later date, as the system is designed to accommodate other functionalities depending on use case.

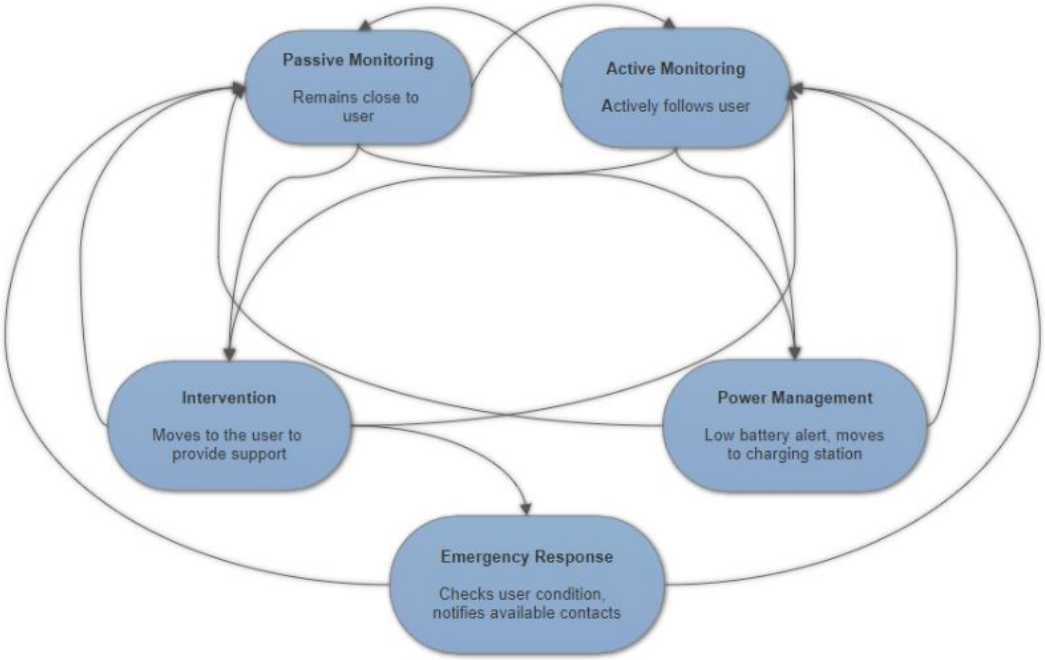

Figure 3.1: State Transition Diagram. Shows interactions between possible protocol states (excluding initial setup state which is a single-occurrence state)

3.1 Monitoring

System procedure that mandates the robot stay near to the person at all times. The robot stays within approximately three strides of the person at all times. It is designed

to accommodate the user as much as possible while ensuring their safety. Passive monitoring is the default robot state, i.e. from here the robot will move to all other states.

In these states, the robot will observe the person's movement and gait to determine if they are at risk of falling. The kinect will communicate skeleton and color data to the robot, which will make this determination. In the event that the person is seen to be falling, the robot will transition to the emergency response state, and in the event that the robot is getting low on battery it will transition to the power management state, as shown in the following sequence diagram.

3.2 Intervention and Response

When a person appears to be falling, the robot transitions to intervention. This state can be reached from passive monitoring state or active monitoring state. This state may call passive monitoring state, active monitoring state, or emergency response state. If the fall is not interrupted, the robot transitions to emergency response. This state can be reached from intervention state. This state may call passive monitoring state or active monitoring state.

This state is active when the user appears to be falling. In this state, the robot will communicate with the stepper to quickly move to provide support to the person. As it reaches the person, the stepper will raise an arm to provide support. The robot will then ask if the person is okay. If the fall is interrupted and the person responds that they are okay, the robot will move to active or passive monitoring. If fall is not interrupted the robot will move to emergency response state.

This state is active when the user has fallen. In this state, the robot will attempt to analyze the user's condition. If contact can be made with the user or the user responds that they are not okay, robot will call for assistance. Additionally if contact cannot be made with the user the robot will record their condition via video, and call the emergency contact using VOIP. The contact's information will be sent to the GUI to be displayed. When the person arrives, they will be prompted to indicate that the person is okay, resetting the robot to monitoring mode.

3.3 Initial Setup

When the robot arrives to the user, the technician will take the user through the below steps to setup the robot. This state is active when the robot arrives and it turned on for the first time. The user will be asked to enter an emergency contact. The user will be asked to enter some kind of unique identification. The user will be asked to identify a static charging station. The robot will then enter passive monitoring state.

3.4 Power Management

System procedure that activates when the robot is about to run out of power. This state can be reached from any monitoring state. This state is active when the robot is

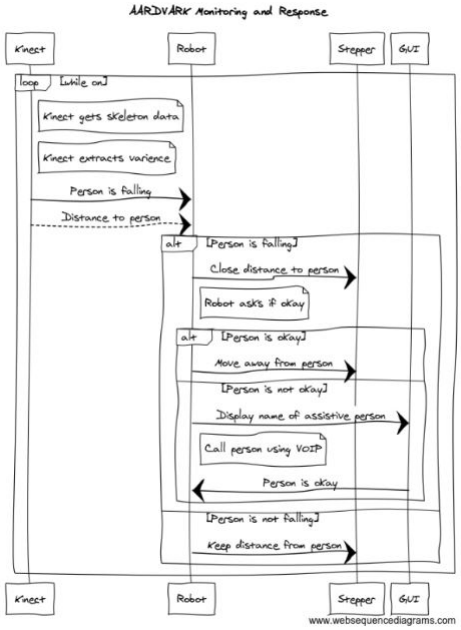

Figure 3.2: Sequence diagram showing interaction between Kinect, Robot, Stepper, and GUI during monitoring, fall intervention, and fall response.

about to run out of battery. When the battery is getting low, the robot will verbally alert the user of the fact. If the user responds, the robot will move to a charging station to change batteries. If the user does not respond, the robot will revert to it's previous state and continue to alert the user every ten minutes, then every minute when the battery level becomes critical.

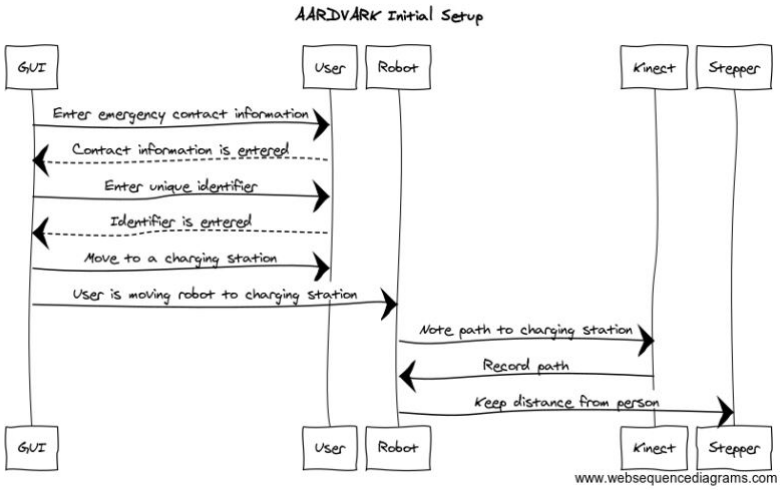

Figure 3.3: Sequence diagram showing interaction between Kinect, Robot, Stepper, GUI, and User during initial setup.

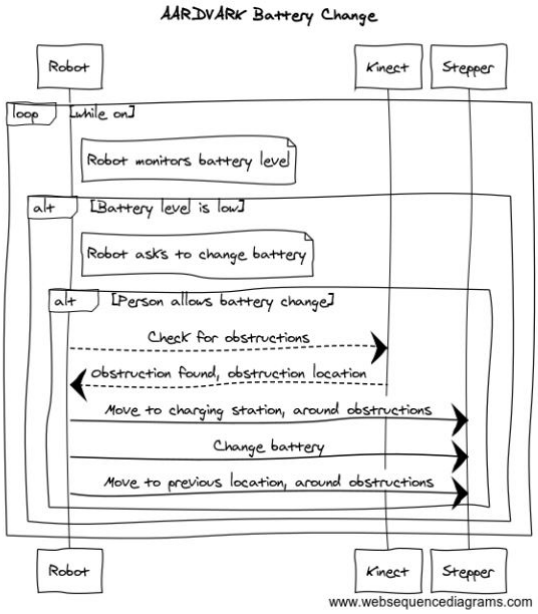

Figure 3.4: Sequence diagram showing interaction between Kinect, Robot, and stepper during power management.

4 Data Design

4.1 Data Description

The system takes in information about the user's body position in the form of coordinates of points on their body. These are stored in a 3D array which holds position data over a time period, with each frame's information added along the third dimension. When the array is full, the variances of each point are calculated over the array. These variances are passed to a random forest classifier which has been built on training data taken by developers to mimic normal and extreme amounts of sway while walking.

The classifier then outputs a boolean value representing whether the person's movements indicate normal movement or a fall risk. That value, along with the person's most recent distance from the robot, is passed to the robot's OS and is translated to a command to be executed. If the person has normal movement, the robot will either remain still or move with the person. If the movement is identified as pre-fall, the robot will approach the human and offer assistance.

5 Component Design

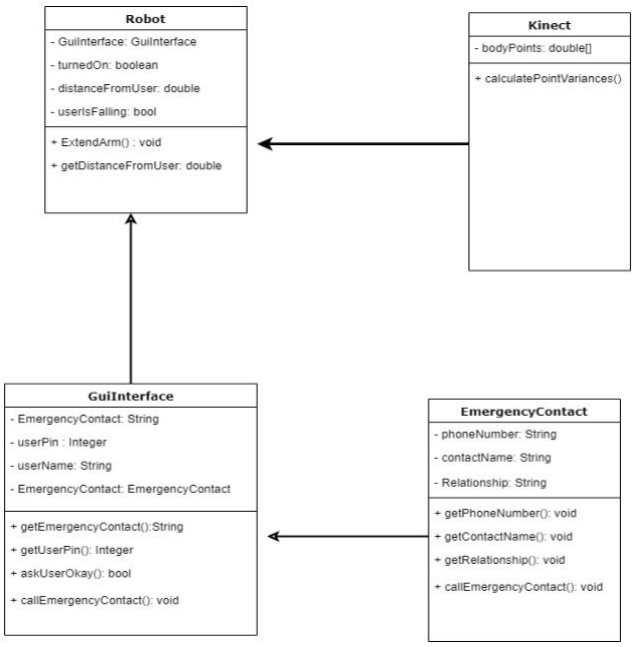

Figure 5.1: Class diagram of relevant system classes. <Change this>

5.1 Kinect

Stand-alone hardware that captures color data and skeleton data for visible humans. skeleton data is represented as a collection of keypoints on the body in 3D space. All code is written in C#.

Methods:
calculatePointVariances(): Used to calculate point variances from data collected (user body points)

5.2 Robot

Hardware and software consisting of the 'brain' of the robot and the motor of the robot (elsewhere referred to as the 'stepper'). Brain of the robot will run on a desktop or mobile pc attached to the body of the robot. Robot will use incoming kinect data to estimate distance from person, perform path finding, and determine when the person is falling. When the person is falling, the steppers will activate and the robot will move to the person to provide support. All code is written in Python

Methods:

ExtendArm(): Extends the arm of the robot to catch user, is called when user fall is detected.

getDistanceFromUser(): Getter method that returns the distance from the user as a double, is called repeatedly while robot is in passive and active monitoring state.

5.3 GuiInterface

Interface on the robot through which the user or emergency contact will communicate with the robot. Stores some basic data about the person and emergency contact. Will also facilitate checking if the person is okay when a fall is detected. All code is written in Python

Methods:

getEmergencyContact(): Getter that return the name of the user's emergency contact as a string

getUserPin(): Getter that returns the user's pin as an integer

askUserOkay(): Robot asks user if they are okay and depending on the user's response the method will return either true or false

callEmergencyContact(): Call's user's emergency contact by using the getEmergency-Contact getter method.

5.4 EmergencyContact

Subsystem that activates when the robot determines it is necessary to call the emergency contact. Will happen when a person is detected to have fallen and is not responding or has responded that they are not okay. Embodies a VOIP application to facilitate communication with emergency contact over wifi. All code is written in Python

Methods:

getPhoneNumber(): Getter that returns the phone number of the emergency contact as an integer

getContactName(): Getter that returns the name of the emergency contact as a string

getRelationship(): Getter that returns the relationship of the emergency contact as a string

callEmergencyContact(): Calls emergency contact using the information/instance vari-

ables within the class (Phone number)

6 Human Interface Design

Figure 6.1: Setup page

6.1 Overview of User Interface

The user will begin by setting up an account. In the set up, the user will put in their user ID (the suggested format would simply be their first name) and a passcode. The passcode is entered twice in order to make sure the user didn't make any errors while typing in their desired passcode. Alongside this is the information for an emergency contact. The user will need to enter the contact's name, relationship to the user, and number. This is stored for the emergency response. After this, the GUI simply displays the AARDVARK name. The user does not have to interact with the screen outside of the occasional login. The last interface is the emergency response. This appears when the robot has entered its emergency response state. A third party can use this screen to either call for more assistance or to reset the state. Once the system is rest, it returns to the AARDVARK name.

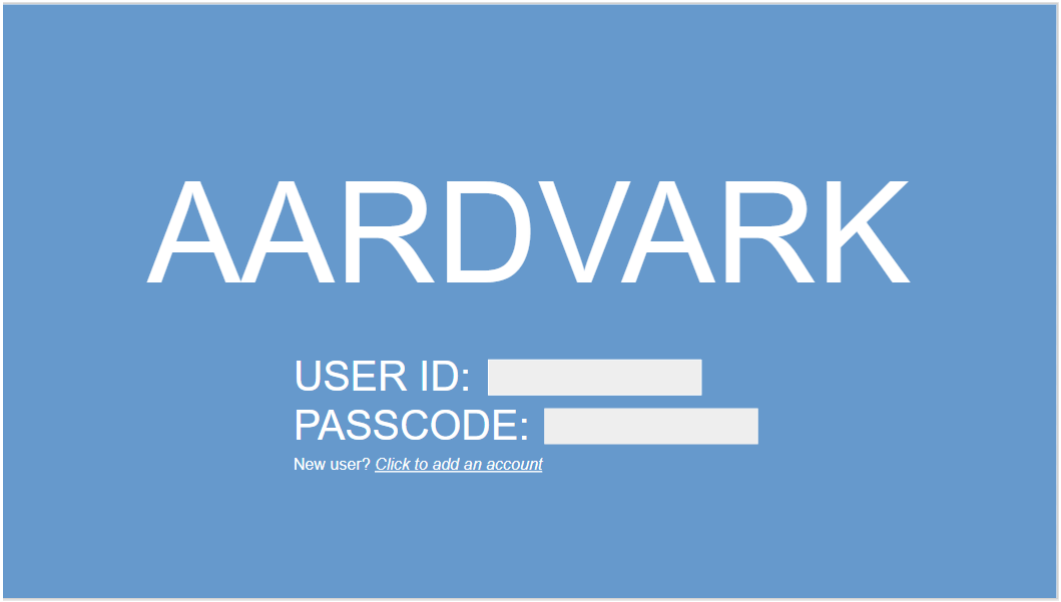

Figure 6.2: Login screen

6.2 Screen Objects and Actions

The GUI is very simplistic and only uses buttons and text inserts. The text inserts are used to determine user credentials and the setup information. Outside of that, all the buttons are used to move from one page to another.

Figure 6.3: Screen that shows under regular functionality

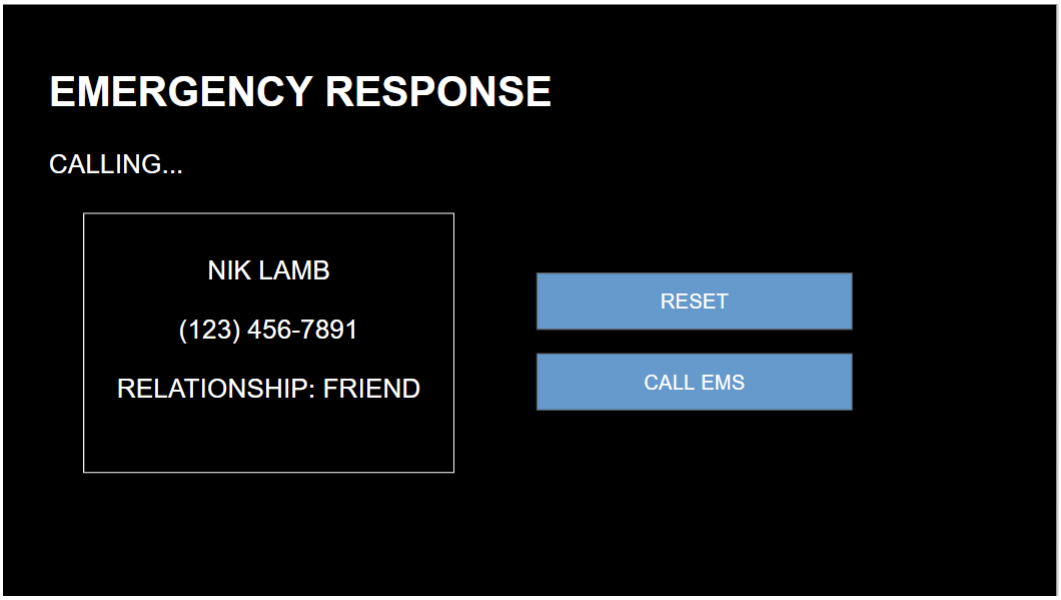

Figure 6.4: Emergency response

7 Requirements Matrix

The following is a table that includes specific states that the robot will be in and also functions that are invoked by the states. The top row contains each of the six states and each individual column contains the functions that are called while the robot is in that state. Several of the functions are invoked by multiple states, mostly due to many states having similar functionality. Note that the class of which the following functions are contained within can be found in the component diagram (Section 5).

Passive Monitoring	getDistanceFromUser()	calculatePointVariances()
Active Monitoring	getDistanceFromUser()	CalculatePointVariances()
Intervention	ExtendArm()	askUserOkay()
Emergency Response	getEmergencyContact()	callEmergencyContact()
Power Management	getBatteryLevel()	alertUserLowBattery()
Initial Setup	EmergencyContact()	getEmergencyContact()