Software Requirements Document for Naturalistic Mobility

TEAM: Naturalistic Mobility (Group 2)

AUTHORS: Dino Kapic, Ryan Ledbetter, Ryan Leska, Cooper McKee, Kerwn Shaw

1. REWRITE SECTIONS IN THIS DOCUMENT WITH YOUR OWN DESCRIPTIONS
2. OMIT SECTIONS MARKED OMIT

Version	Date	Author	Change
0.1		SM	

Table of Contents

1 Introduction....3
1.1 Purpose....3
1.2 Scope....3
1.3 Definitions, acronyms, abbreviations....3
1.4 References....4
2 Overall Description....5
2.1 Product Perspective....5
2.2 Product functions....6
2.3 User characteristics....8
2.4 Constraints....8
2.5 Assumptions and Dependencies....8
3 Specific Requirements....9
3.1 FEATURES....9
3.2 Performance requirements....9
3.3 Design Constraints....9
3.4 Software System Attributes....9
3.5 Other Requirements....10

1 Introduction

1.1 Purpose

The purpose of this document is to establish how the application should interact with the end user, and establish all application requirements functional, and non functional. Once finalized, this document will state what must be accomplished for the application to be considered finished.

1.2 Scope

This SRS covers a number of potential use cases that users may encounter, as well as an overview of the project and its intended uses. It also includes information on the project's UI sketches, but the primary purpose is to give detailed descriptions of anticipated use cases.

1.3 Definitions, acronyms, abbreviations

Term	Description
T1: ROS	Robot Operating System (ROS) is a flexible framework designed for developing robotic applications. Despite its name, ROS is not an actual operating system but a middleware that provides tools, libraries, and conventions for building complex and scalable robot software systems.
T2: Pose Estimation	Pose estimation is the process of determining the position and orientation of an object or person in space, typically using images or sensor data. It is commonly applied in robotics, augmented reality, and computer vision tasks such as tracking movements or recognizing objects in 3D. Techniques often involve machine learning or geometry-based algorithms to infer key points and construct the pose.
T3: Wake Words	Wake words are specific phrases or keywords used to activate voice-activated devices, prompting them to start listening for further commands. Examples include "Hey Siri," "Alexa," or "OK Google." They are typically processed locally using low-power algorithms to detect the phrase before engaging the main system for voice recognition.
T4: Go2	Go2 the Robot Dog is a quadruped robot developed by Unitree Robotics, designed for both commercial and personal use. It is a highly versatile robotic

	dog that can navigate various terrains, carry loads, and perform tasks autonomously or via remote control. Go2 features advanced sensors for obstacle avoidance, AI capabilities for intelligent decision-making, and options for customization with additional modules, making it suitable for applications in inspection, exploration, and entertainment.

1.4 References

[NONE]

2 Overall Description

There has been a ton of recent development in humanoid robots, however, these robots require a ton of complex commands to be entered if you are wanting to run the robot in real-time. The purpose of this project is to allow for a simpler controller interface for users to interact with. There are two controllers that were implemented:

- Visual: Constant video feed that uses pose estimation [T2] to generate commands to send to the robot being controlled.
- Audio: Using wake words [T3] to initialize the program into listening mode and feeding natural language to generate commands.

After these controllers generate the commands, the commands will then be sent to the ROS [T1] to control the robot.

2.1 Product Perspective

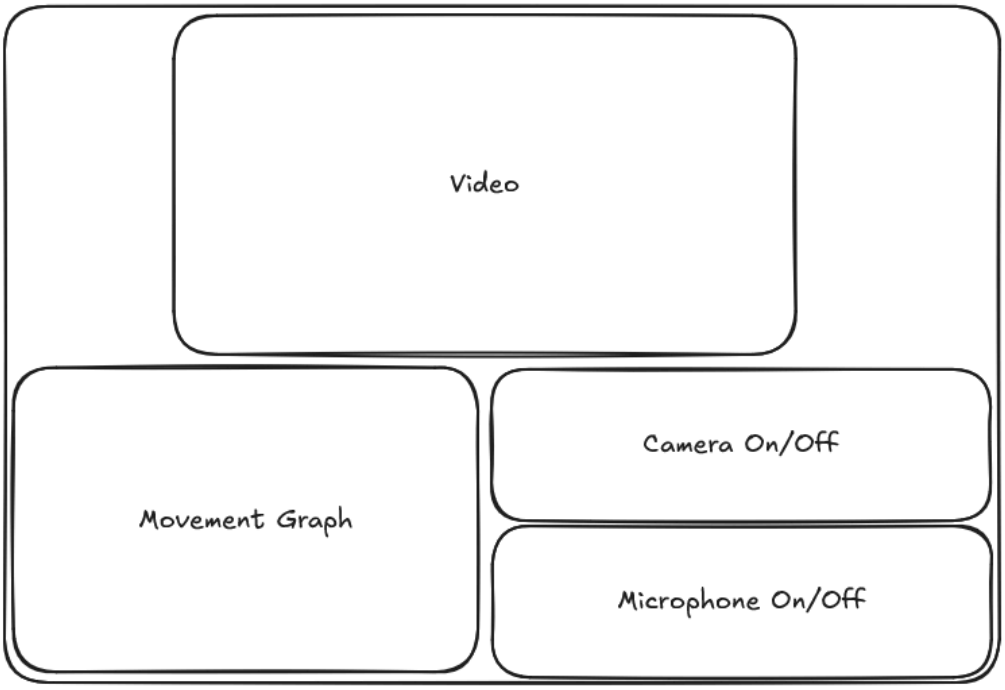
Naturalistic Mobility Controllers: A Python application that uses camera and microphone input to control a robot, currently developed for Go2 [T4].

2.1.1 Concept of Operations

This project is a Python application that runs on Python version 3.10 and is made to be compatible with Ubuntu 20. The application features camera and microphone input from the machine or devices plugged into the inputs. Both features will be able to be turned on and off via the UI. Users will be able to use their arms via the camera input to have Go2 [T4] move forward, backward, and rotate left and right. Users will also be able to use wake words [T3] and directions via the microphone to generate the same commands as the camera.

This project also uses multi-threading to run the main interface along with the camera and audio inputs. By using the multi-threading approach, users will be able to run both controllers simultaneously while generating commands to publish to the ROS [T1 or outputs CSV to move the robot in real-time or simulated real time.

2.1.2 Major User Interfaces



The entire application runs off of one screen as sketched above. The video box displays the camera output, the movement graph displays a visualization of the commands being sent to the output csv, and the camera and microphone buttons turn the audio and visual classification threads on and off.

2.1.3 Hardware Interfaces

This application is designed to run on an Ubuntu 20 linux machine with Python and all of the requirements listed in the requirements.txt.

2.2 PRODUCT FUNCTIONS

All of the functionality is controlled from the GUI with buttons for the camera and audio. When the camera or audio on button is pressed, the respective visual or audio thread is started. If the camera

or audio off button is pressed, the respective visual or audio thread is terminated. Functionality of the audio and visual threads are detailed below.

2.2.1 Audio

The audio thread uses a mix of OpenWakeWord, OpenAI's Whisper and PocketSphinx. OpenWakeWord starts during the initialization of the audio thread and waits until a custom wake word is detected. Once this word is detected, the audio thread runs the main audio recognition function which listens for a user defined amount of time. This audio data is then passed to the Whisper which runs its trained models against the audio data to determine what is being said. This outputs a string and a confidence score. If that confidence score is below a threshold, it will call PocketSphinx to also identify the audio. These strings are then passed to our command searcher. Our command library is built using synonyms of our commands in which the user can rate these with a confidence for similarity. If a command or synonym is found within either string, we then add that command to the command queue

2.2.2 Visual

The visual thread uses Google's MediaPipe and the Python cv2 package to run the main components of the visual controller. The Python cv2 package allows for our application to quickly detect the default camera on any machine and will not require a user to input what camera they would like to use. MediaPipe is a pose estimator that feeds 32 landmark points by default but allows for customization to delete or add landmark points to the frame. MediaPipe by default processes on every frame, however, for this application, we only process the pose classification on ~20 frames. This application only uses the landmarks for wrist, shoulder, and elbow points on both the left and right arm. To classify the pose, the camera controller will use the Python math library to determine the angle dependent on the shoulder point of each arm and then process the estimated pose through a binary classifier to validate the pose.

2.3 User characteristics

The typical user for this application will be a researcher in Dr. Bowen Weng's robotics research lab at Iowa State University. This application is designed to have only one user at a time and work in a local environment.

2.4 Constraints

The application must run on Ubuntu 20 and use only a CPU for processing.

2.5 Assumptions and Dependencies

It is assumed that the computer will have a USB camera and microphone as well as internet access for installing pip dependencies.

3 Specific Requirements

3.1 FEATURES

3.1.1 Main GUI for user interaction with the application

3.1.2 Ability to launch and terminate audio and visual threads from the GUI

3.1.3 Ability to view camera output from vision thread on GUI

3.1.4 Ability to see movement commands being sent to ROS out output CSV

3.1.5 Audio classification for controlling the robot with audible commands

3.1.6 Visual classification for controlling the robot with visual gestures

3.2 Performance requirements

This application is designed to be run at 10 hz or send a new movement command every 0.1 seconds.

3.3 Design Constraints

This application must be designed to work on an Ubuntu 20 device and use a usb camera and microphone for input.

3.4 Software System Attributes

3.4.1 Reliability

Our solution provides a high level of reliability by continuous confidence and error checking. When an error occurs the system is able to either handle the error or restart the subsystem in which the error originated. To maintain reliability in our output, inputs are rated with a confidence level such that lower confidence commands are ignored preventing unreliable usage.

3.4.2 Availability

Our solution will always be available as long as the system it is running on meets the requirements. Due to our solution being a local one, network outages will not affect our users. When an error may occur, the system is able to handle the error or due to the threaded nature of the processes, an individual thread can be refreshed to maintain workability.

3.4.3 Security

Our solution provides a high level of security by running everything locally. A local solution allows for the user to maintain complete control of their data and privacy. All models that are built for our solution are able to be kept secure and private to an individual user and their use case.

3.4.4 Maintainability

Our solution is structured in a very modular fashion. Following the Python Application-Module Structure allowed us to partition our applications subsystems such that finding where and how to make changes as well as fix errors is quick and easy. Along with the structure, we have worked with our client to provide resources so he is able to continue this project with ease.

3.4.5 Portability

Our solution is not highly portable but this is by design. We were tasked with creating an application to run on very specific hardware. Our solution is catered to run specifically in a lab environment at Iowa State University so must adhere to certain security and hardware criteria. The requirements were first met with portability coming second to functionality.

3.5 Other Requirements