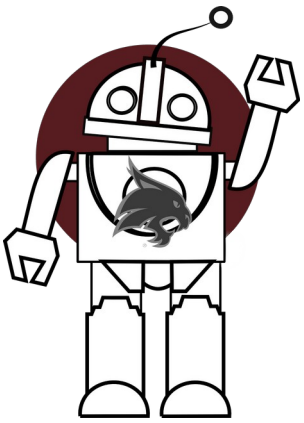
Bobcat Robotics

Software Requirements Document

Software Name: Multi-Robo Guider (MRG)
Author: Gentry Atkinson
Version: 1.0.0



1. Introduction

Product Purpose: this product is being developed as a generalized robotic guidance platform. This product will take guidance input from the console and translate those values into positional data for a robotic platform.

2. Standards

Hardware: this product should be capable of running on a Linux platform with the gcc compiler and a means of text I/O.

Schedule: completion of this product is expected no later than 09 June.

Language: this project will be implemented using the C++ programming language.

3. System Description

System Context: this will be a self-contained piece of software that will not rely on external platforms for its basic operation.

User Characteristics: users will be trained robot operators who are proficient with computer operation. This project will not be responsible for user training.

4. Functional Requirements

4.1 Startup: when the software starts, it should ask the user for a number of robots to track. It should also ask the user for a unique identifier for each robot. You can assume that the user is entering unique names for each robot, you do not need to check.

4.2 Menu: the user should be shown the menu of possible commands and should be able to select one command from the menu. Your program should print an error message if the user enters a letter that is not in the menu. The following commands are available in MRG:

- **M**- move one robot
- **D**- print the distance each robot has moved
- **Q**- quit the program

4.3 Move: The user should be prompted to enter the unique identifier of one robot and a direction. The program should print an error if the user enters an invalid direction or if they enter an invalid robot identifier. The following directions are allowed:

- **U**- up or positive y
- **D**- down or negative y
- **R**- right or positive x
- **L**- left or negative x

Your program should update the position of the correct robot by an amount equal to the robots speed (defined in **Section 5.1**) and then print the new position. Every robot starts at position 0,0 and can move infinitely in any direction. The position should always be printed as **X,Y**.

4.4 Distance: This menu option should print the unique identifier of each robot and the distance that it has moved in neatly formatted columns.

4.5 Quit: Your program should print a short parting message when the user selects ‘Q’.

5. Non-functional Requirements

- The program should be submitted as a .cpp file named [Author’s LastName]_assignment1.cpp, with [Author’s Last Name] replaced by the author’s last name.
- The first three lines of the program should be the author’s name, the date that the assignment was written, and a list of collaborators.
- The author is expected to follow the Style Guidelines.
- Your program should store all information about Robots in an array of Robot structs (described below). The size of this array is given by the user at startup (described in **Section 4.1**). Your array should not be any larger than necessary.
- Your functions should be well commented. The comments for each function should include:
  - Name
  - Parameters
  - Returns
  - Side Effects

5.1 Robot struct
The program should store the robot's name, speed, and positional data in a struct with the following members:

- X- the current X value of the robot's position
- Y- the current Y value of the robot's position
- lastCommand- the last direction that the robot moved
- currentSpeed- the speed that the robot is traveling
- distance- the total distance that the robot has traveled
- name- the name of the robot

The initial values of the X and Y members should be set to 0. This could be done with a **constructor** if that's familiar or in the **main** function.

A robot speeds up the longer it moves in one direction. A robot's first move in a direction will move one unit in that direction, the second move in the same direction increases the robot's speed to two, the third move in the same direction to three, and then to four. Four is the maximum speed for these robots. Changing direction resets the robot's speed to 1.

The distance value should store the total number of squares that the robot has moved, not its euclidean distance from the starting position. For example, a robot that moves four spaces up and then three spaces right should have a distance of seven even though the straight line distance back to (0,0) is only five.

5.2 findRobot function
This function should find the index of one Robot in the array of Robots using its unique identifier.

The parameters of this function are:

- Robot[] roboList: a list of robots
- string identifier: the target unique identifier
- SIZE: the number of robots in roboList

Return the index of the robot with a matching identifier as an int, or -1 if the identifier is not in the list.

5.3 moveRobot function
The position of the robot struct should be updated using a void function called moveRobot.

The parameters of this function are:

- Robot r: a Robot passed by reference
- char d: a character value representing one direction

The robot's position should be updated as described **in Section 4.3** (so if d == 'U', then increase r.y by r.currentSpeed, etc.)

This function also needs to update the current speed and the distance traveled for the robot. The robot's speed increases by one unit every time it moves in the same direction up to a maximum speed of four. See **Section 5.1** for more information about speed and distance.

Only this function should be used to change the robot's position after the initial position has been set.

5.3 Interface: the display should be neatly formatted and easily readable by the user. You may choose the exact formatting of the output, but it must be consistent on a terminal with an 80 character width.

Your program should let users enter characters as upper OR lower case.

6. Guidance

- The robot should speed up every time it moves in the same direction. You need to be able to compare characters regardless of whether they are upper or lower case. The ctype.h library might help.
- The program should keep running until the user selects Q. There should be a loop in your main function that breaks when a user types Q.
- The iomanip library might help you with your formatting, but it is not required. Just make sure everything looks good.
- You will need a search algorithm to find one Robot in the list of Robots. Think about which one is best for this project.

6. Sample Test Cases

Input	**Output**

Start program
3
Fatemeh
Ariel
Carlotta
	Welcome to MultiRobo Guider.
	Please select:
	  M- move
	  D- distance
	  Q- quit

M Fatemeh u	Fatemeh’s position is 0,1
m Ariel R	Ariel’s position is 1,0
K	Invalid command
m Carl R	Robot not in list.
M Fatemeh f	Invalid direction.
m Fatemeh U	Fatemeh’s position is 0,3
D	Fatemeh 3 Ariel 1 Carlotta 0
Q	Goodbye