Software Document

Project: LIBERTY

Task: SOFTWARE ARCHITECTURE

Document Version Number: 1.0
Date: 2017/11/20
Author: Bill Zhang
Editor: Andi-Camille Bakti
Edit History: https://github.com/Gabetn/DPM_01_Project_Documentation

TABLE OF CONTENTS

TABLE OF CONTENTS **2**

1.0 Software Architectural Diagram **3**

2.0 Architectural Description & Rationale **4**

3.0 Architectural Components **4**

1.0 Software Architectural Diagram

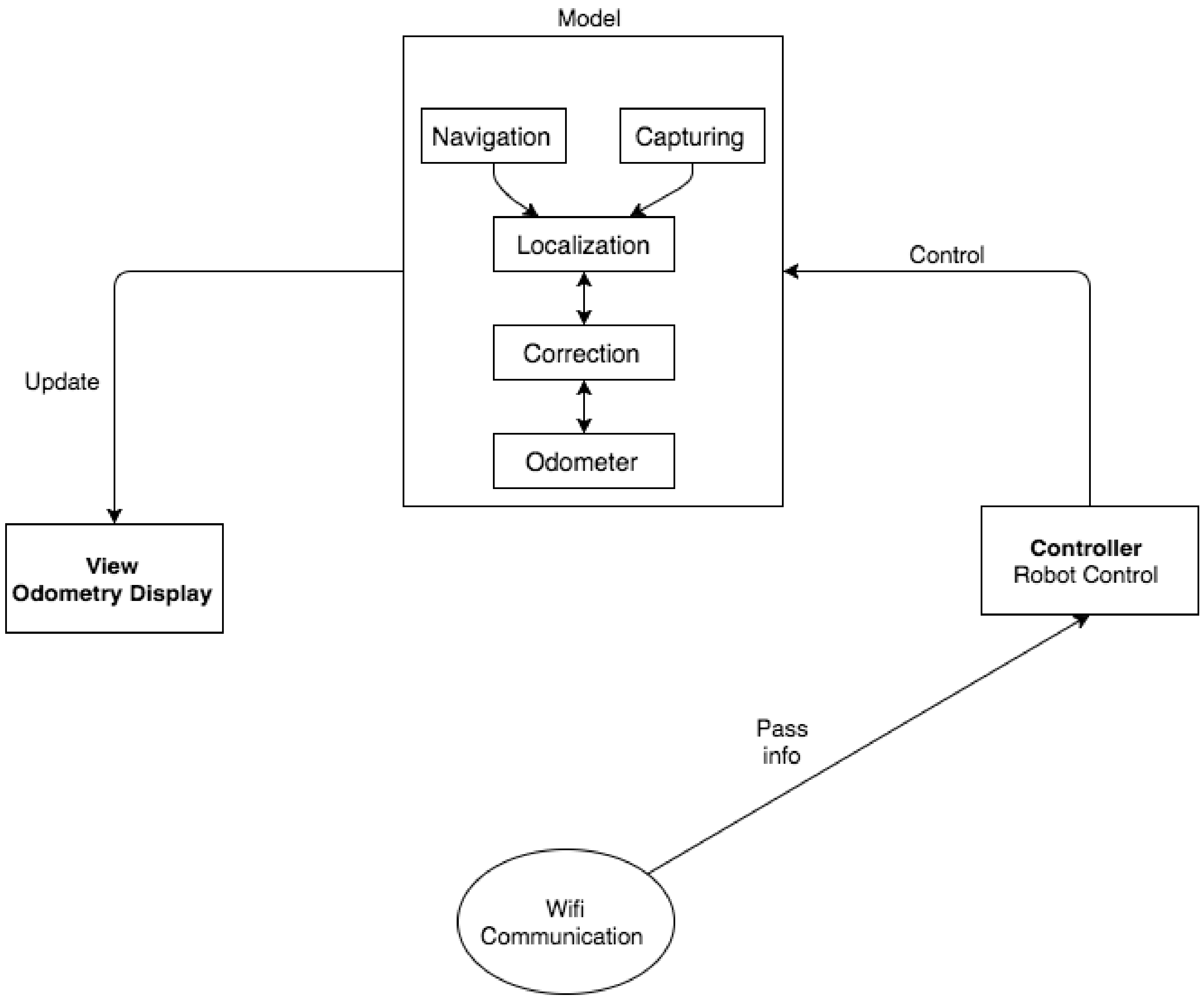

Figure 1: Diagram of the system's software architecture.

2.0 Architectural Description & Rationale

A hybrid version of Model-View-Controller and Layered Architecture is used in our design. Although there is no requirement to represent data in multiple ways, MVC is selected as the overall architectural style because it is widely used in the previous labs and a controller is always required to call different classes. Therefore, MVC helps to make the overall structure of the code very clean, and it allows us to update the functional classes without changing the controller. Also, MVC provides a better solution to present data. It is well understood that there is no need to represent any data during the competition. However, having necessary information such as odometry data or way point marks displayed on the screen is very helpful for debugging purposes. Therefore, a minor reason to select MVC is that it offers a decent way to display the data that is required for debugging purposes.

A layered architecture is selected to organize the subsystems in the model component with the bottom layer being the odometer and the top layer being the functional classes. The functionality of the system must be organized into a layered style because navigation and capturing depend on it location which is determined and corrected by localization and correction. The localization and correction are based on the odometer, therefore, the relation of their dependency determines that the odometer is in the bottom and functionality that is dependent is at the top. The selection of the layered architecture allows us to add and update the functional classes without changing the bottom layer. For example, navigation and capturing are two new classes written inclusively for the project, therefore they are the only classes that are not tested by labs. So they need to be updated and changed constantly. With the layered architecture, it is possible that we can update and change the navigation and capturing without changing the layers beneath them. The workload for updating is minimized in this way.

3.0 Architectural Components

The controller component in the architectural design corresponds to the robotControl class in the system. The model component corresponds to all the classes in the system except robotControl and odometry display. The view component corresponds to the odometry display class in the system. This class might be removed after debugging to reduce the size of the system.