**Software Formalization**

**Year:** 2024 **Semester:** Spring **Team:** 5  **Project:** Dodgebot

**Creation Date:** 02/15/24 **Last Modified:** 02/17/24

**Author:** Maximilian Drach **Email:** [mdrach@purdue.edu](mailto:mdrach@purdue.edu) ([mdrach100@yahoo.com](mailto:mdrach100@yahoo.com))

**1.0 Utilization of Third-Party Software**

|  |  |  |  |
| --- | --- | --- | --- |
| Software | Description | Purpose | Licensing |
| STM32CubeIDE | A UI application for configuring STM microcontrollers. | Programming and testing GPIO pins via a user interface. | Free |
| AutoDesk | CAD design software | Designing motor enclosures for the robot. | Student license provided by Purdue. |
| Eagle | PCB design software | Designing PCB for communication and power systems. | Student license provided by Purdue. |
| VSCode | A free IDE for writing code. | Developing software components and algorithms. | Free |

**2.0 Description of Software Components**

The Dodgebot has 5 major components: user interface, image processing, object tracking, dodging algorithm, and motor control.

User Interface (UI):

* Development: Entirely built by our team using VS Code in Java.
* Platform: Runs on Dell XPS 9710 with an Intel I7-11800H and Nvidia RTX 3060 (60 W).
* Functionality: This interface handles displaying and collecting information for the user. The UI collects the users’ punch height (meters), the amount of time the user wants to play (minutes: seconds), and the start button which initiates the Dodgebot’s reaction system. The UI displays to the user the amount of time left to play before deactivating the Dodgebot’s reaction system and a non-emergency stop button.

Image Processing:

* Development: Entirely built by our team using VS Code in Python with NumPy and OpenCV libraries.
* Platform: Runs on Dell XPS 9710 with an Intel I7-11800H and Nvidia RTX 3060 (60 W). Processes data from an ELP-USBFHD08S-L29 camera over USB.
* Functionality: This program is responsible for the pre-processing of the original raw frame into a usable undistorted square frame. First, the distorted frame is sent over USB from the camera to the computer. Next, utilizing pre-derived empirical intrinsic and extrinsic weight matrices from Zhang’s calibration method [2] the frame is undistorted to an ideal pinhole camera frame. Finally, the frame is cropped into the center thus converting the 640x360 frame into 360x360 resolution.

Object Tracking:

* Development: Entirely built by our team using VS Code in Python with NumPy, Kmeans\_pytorch, and OpenCV libraries.
* Platform: Runs on Dell XPS 9710 with an Intel I7-11800H and Nvidia RTX 3060 (60 W).
* Functionality: Responsible for utilizing the user input and processed image to track the real-space coordinate locations of the gloves (punches) and punch bag (Dodgebot). First, the pixels are converted to the HSV color space and then parsed to discern if any of the pixels match the color profile we are trying to track. The pixels that match the color profiles are then mapped to a real-space coordinate given the user’s input punch height using the P2C algorithm [1]. The real-space coordinates are then passed into either a punch or robot tracking processes that run in parallel.
  + The punch tracking utilizes K-Means to distinguish the two different punch coordinates and passes in the previous punch coordinate center to consistently track the same punch. The main punch that the robot is reacting to is deciding by comparing the L2 distance between the cluster centers and the robot center. The cluster with the smaller distance is considered the “main punch” which is then passed to the Dodging Algorithm.
  + The robot tracking averages the collection of points to find the point center. It then approximates the height of the robot using the location using the DP2C algorithm [1] and finally converts the pixel location to the real-coordinate location using the P2C algorithm [1].

Dodging Algorithm:

* Development: Entirely built by our team using VS Code in Python with NumPy library.
* Platform: Runs on Dell XPS 9710 with an Intel I7-11800H and Nvidia RTX 3060 (60 W).
* Functionality: Responsible for deciding the ideal location to dodge an incoming punch. Utilizes the Perpendicular Dodging Algorithm (PDA) [1] to determine Dodgebot's dodging direction.

Motor Control:

* Development: Entirely built by our team using VS Code in C and flashed onto the STM32F207VG.
* Platform: STM32F207VG microcontroller configured using STM32CubeIDE.
* Functionality: Receives rotational angle data via USB-UART. Converts and normalizes angle difference for proportional pulse position modulation. Controls Dodgebot's motor movements with the pulse width modulation.

**3.0 Testing Plan**

User Interface:

* Priority: 3 (Medium)
* Test the UI systems by outputting the user input to the terminal and checking to see if the output values match the input given to the UI system.
* Also, check to see if the non-emergency stops button and the end of the timer outputs an interrupt to the terminal.

Image Processing:

* Priority: 2 (High)
* Test if the system undistorts a checkerboard at the full resolution (640x360) by seeing if the checkboard looks like it was taken with a flat frame pin-hole camera. The reprojection error should be less than 1cm, which is the standard parameter error we are aiming for.
* Test the square algorithm by seeing if a center square is in the same location within the center of the frame when the resolution is changed from a full frame to 360x360.

Object Tracking:

* Priority: 1 (Critical)
* Test our punch tracking systems by pre-measuring two real coordinate punch locations and comparing the output simulated real locations from the punch tracking algorithm.
* Will also track the system’s recognition of the incoming “main punch” by throwing punches at the expected target location and seeing which hand the computer associates with the “main punch”.
* Involves 20 consecutive testing trials in locations all over the tracking space, with at least 12 trials near the edge of the frame where distortion is the highest.
  + The simulated punch start location should be within 1cm of the measure start location.
  + We will track the system’s recognition of the incoming “main punch” by throwing punches at the expected robot location and seeing which hand the computer associates with the “main punch” (marked with a red circle).
  + If the start location is within 1cm tolerance and the “main punch” is always marked with a red circle throughout the duration of the jab, then this is considered a passing trial.
* Will test out robot tracking by pre-measuring the robot's z-coordinate space height and comparing the simulated z-coordinate height from the DP2C algorithm [1]. The threshold is 5% of the radius length of the robot’s height. If the z-simulated height is within the threshold of the real z-coordinate height, then it passes the test.

Motor Control:

* Priority: 1 (Critical)
* Analog Testing:
  + Involves passing a simulated motor control angle to the microcontroller from the computer over a USB-UART interface, testing the output from the GPIO control pins on an oscilloscope, and measuring the difference between the expected pulse width and the output pulse width from the microcontroller.
  + The next step is to measure the signal delay between when new rotational angles are sent from the computer to when the pulse width modulations appear on the oscilloscope.
  + Measure the signal delay between the motor driver's rotational position data and the Dell XPS 9710 updated angles.
* Physical Testing:
  + After analog testing, use protractors to measure the initial angles on the robot arm and input a test rotational angle. Next, compare the new robot rotational angle with the expected results from the input angle change.

Dodging Algorithm:

* Priority: 1 (Critical)
* Virtual Testing:
  + Outputs a captured frame with a circle to represent the simulated robot on the Dell XPS 9710. Track the circle’s movement around the frame to show where its simulated location would be given an input punch.
  + While this testing can test the general effectiveness, it is limited to the small frame size (360x360) and a delayed 60Hz display.
* Physical Testing:
  + Involves the entire Dodgebot pipeline and tests the effectiveness of the dodging algorithm in the real world. This testing can only be implemented after all the systems have passed their system test and the robot is safe to operate around.

**4.0 Sources Cited:**

[1] M. Drach, “Maximilian Drach ECE477 Journal Report,” <https://engineering.purdue.edu/477grp5/>. https://engineering.purdue.edu/477grp5/Team/progress/progress3.html (accessed Feb. 15, 2024).

[2] Z. Zhang, “A flexible new technique for camera calibration,” IEEE Journals & Magazine | IEEE Xplore, Nov. 01, 2000. https://ieeexplore.ieee.org/document/888718