Object Recognition and Path Smoothing Robot

Jake McKenzie

[jake314@uw.edu](mailto:jake314@uw.edu)

Alex Marlow

[alexmarlow117@gmail.com](mailto:alexmarlow117@gmail.com)

Ammon Dodson

[ammon0@uw.edu](mailto:ammon0@uw.edu)

Prepared: 11/25/2018 Rev: 0.3

|  |  |  |
| --- | --- | --- |
| **Revision History** | | |
| V0.1 | 10/18/2018 | Initial Specification |
| V0.2.0 | 11/01/2018 | Additional information on ROS |
| V0.2.1 | 11/01/2018 | Added ethical concerns |
| V0.2.2 | 11/03/2018 | Modelled the overall system architecture |
| V0.3 | 11/25/2018 | Additions to the introduction. Rewrite of the requirements section with enumeration. |
| V0.3.1 | 11/22/2018 | Additions to the System Architecture and System Design section including hardware architecture and software architecture with higher level and lower level diagrams with text describing these diagrams and text for these sections generally. |

Table of Contents

[1. Introduction 3](#_Toc530519503)

[1.1. Remote Control 3](#_Toc530519504)

[1.2. Autonomous Control 4](#_Toc530519505)

[1.3. Scope 4](#_Toc530519506)

[2. Requirements 4](#_Toc530519507)

[2.1. Remote Control 4](#_Toc530519508)

[2.2. Autonomous Control 5](#_Toc530519509)

[3. System Architecture 5](#_Toc530519510)

[3.1. Robot Operating System (ROS) 5](#_Toc530519511)

[3.2. The McCourt Filter 6](#_Toc530519512)

[3.3. Human Controller 6](#_Toc530519513)

[3.4. TurtleBot3 Burger 6](#_Toc530519514)

[4. System Design 6](#_Toc530519515)

[4.1. Hardware Design 6](#_Toc530519516)

[4.1.1. Objectives 6](#_Toc530519517)

[4.1.2. Constraints 6](#_Toc530519518)

[4.1.3. Composition 8](#_Toc530519519)

[4.1.4. Uses and Interactions 8](#_Toc530519520)

[4.1.5. Interface 8](#_Toc530519521)

[4.1.6. Resources 8](#_Toc530519522)

[4.1.7. Details 8](#_Toc530519523)

[4.2. Software Design 8](#_Toc530519524)

[4.2.1. Objectives 8](#_Toc530519525)

[4.2.2. Constraints 8](#_Toc530519526)

[4.2.3. Composition 8](#_Toc530519527)

[4.2.4. Uses and Interactions 8](#_Toc530519528)

[4.2.5. Interface 8](#_Toc530519529)

[4.2.6. Resources 8](#_Toc530519530)

[4.2.7. Details 8](#_Toc530519531)

[4.3. Human Interface Design 8](#_Toc530519532)

[4.3.1. Functionality from the User’s Perspective 8](#_Toc530519533)

[4.3.2. Interface Objectives 8](#_Toc530519534)

[4.3.3. Interface Constraints 8](#_Toc530519535)

[4.3.4. Use Cases 8](#_Toc530519536)

[5. Ethical Considerations 8](#_Toc530519537)

[6. References 9](#_Toc530519538)

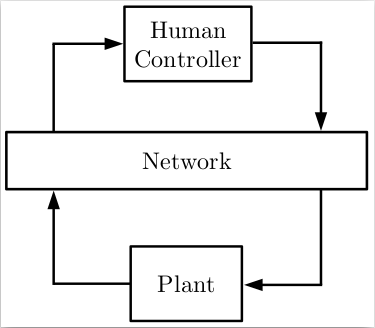
[7. Errata 9](#_Toc530519539)

# Introduction

The Object Recognition and Path Smoothing Robot is systems project with multiple objectives, they are, in sequence:

* Implement for verification Dr. McCourt’s delay mitigation algorithm.
* Demonstrate Simultaneous Localization and Mapping (SLAM)
* Implement a “finder robot” by integrating machine vision with the SLAM functionality

Additional objectives may include:

* Using beacon triangulation and/or GPS to fuse additional location information into the SLAM or finder functionality
* Implement functionality to allow the system to report back on mapped spaces or found objects

## Remote Control

Remote control systems have many useful applications; The typical use case involves using robots to interact with an environment that is too hazardous for a person. Any such remote control involves some delay of both outgoing control signals and incoming sensor data. In some situations, this delay may impair the intended function of the remote system.

Dr. McCourt has developed a set of filters intended to be placed in such a delayed, closed-loop control system. These filters apply a mathematical transformation on both incoming and outgoing loop signals such that communication delays are mitigated. This specification outlines the development of a controller-robot system intended to demonstrate the McCourt filter.

Figure 1: A typical remote-control system

## Autonomous Control

There may also be use cases for robots in hazardous environments where direct human control is impossible. Such a robot must be able to autonomously navigate and interact with an unknown space. SLAM is a fundamental technology for such autonomous activity, allowing the robot to navigate. Additionally, such an autonomous machine must be able to sense and recognize an objective before being able to interact with it. Computer vision is another fundamental technology for sensing a real-world environment. Sensing a condition is a necessary first step for being able to act based on the current environment.

## Scope

This specification covers the following:

* Any and all hardware modifications to the TurtleBot3 Burger
* Software installed on the burger bot insofar as it deviates from the stock installation
* Base station control software setup insofar as it deviates from stock installation
* Any necessary techniques for integrating a Human Interface to the base station installation
* Any methods used to implement a communication delay between the robot and its base station
* Implementation of the McCourt filter
* An overview of implementing SLAM on the TurtleBot3 Burger
* Integration of OpenCV into the TurtleBot3 Burger

# Requirements

In this section we will delve through the minimum requirements of this project to translate the needs of our client Michael McCourt into precise targets, establish metrics for a successful product and support design trade-off decisions. For this revision of the design specification we will articulate the marginally acceptable target values, without any of the ideal target values for our stretch goals.

## Remote Control

#### The Robot used will be the TurtleBot3 Burger

This robot will use the TurtleBot3 Burger as a base. All the hardware and software modules of the remote component must fit within the constraints of that platform, including such things as power, weight, and processing speed.

#### There must be a base station interfaced with a USB game controller

The Human controller we have available currently is a USB game controller. This controller will have to be connected to some base station that can connect to the TurtleBot3 wirelessly.

#### The base station must communicate with the robot through a delayed link

In order to demonstrate the McCourt input-output transformation the robot must be remotely controllable. Control signals must be routed through a delayed communication medium. Ideally, or as a second stage, the human controller’s feedback information should also be routed through the delayed medium.

#### The Robot Should be wirelessly controllable

To fully demonstrate the usefulness of the McCourt filter we should model a real-world situation where the robot is out of sight and controllable.

#### The robot must be able to report its position in space

To fully demonstrate the McCourt input-output transformation the full control loop should be routed through it as shown in Figure 2. This will require the position feedback to be in the form of a simple numerical array such as an x-y coordinate.

#### There must be a means of recording and comparing the planned and actual path the robot follows

To demonstrate the McCourt input-output transformation we need to be able to compare the actual and planned path and have some measure of how closely they align. We should be able to make multiple test runs with and without the filter functioning and compare the course fidelity in aggregate.

## Autonomous Control

#### The Robot should be able to methodically explore an area to its limits

If the robot is being used to search an area it will be important to ensure that the full area has been search. Otherwise the system could produce a false negative. It is also important to consider the orientation and effective range of any sensors to ensure that they have made adequate coverage of the search area.

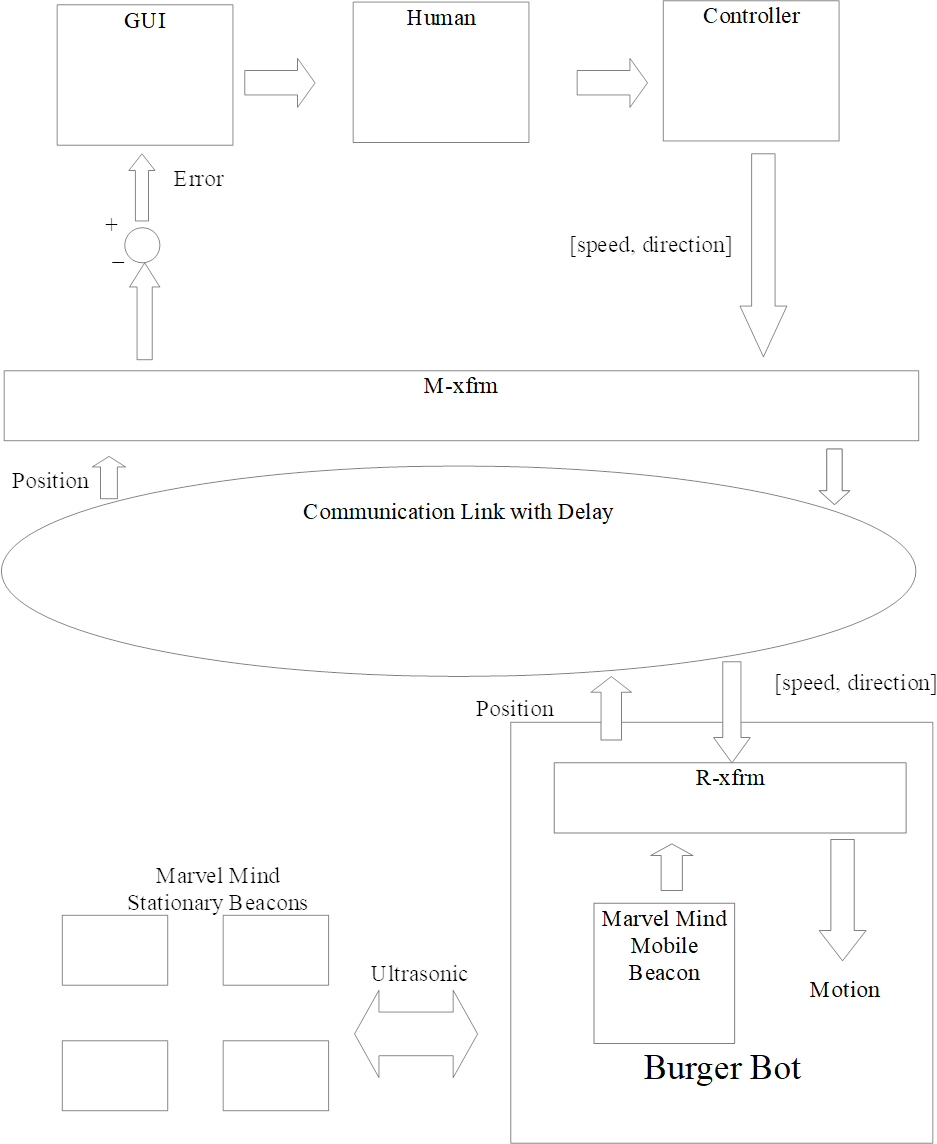
#### The robot should create a map of the area

In order to be useful, The map should available on the base station as quickly as possible. This map should also include any found objects and links to sensor data for those objects.

#### The robot should be able to identify and locate pre-programmed objects

It should also be able to identify, locate, and report on the location or condition of some object in the environment.

# System Architecture

The main job of ORPS-Robot (Object Recognition and Path Smoothing) is to validate the research of Michael McCourt and a scheme for exploring object recognition via OpenCV with Robot Operating System (ROS). In this section we will develop our methodology for the overall system architecture by exploring the flow of data from each component in the overall system, by referring to the figure 2 on our right.

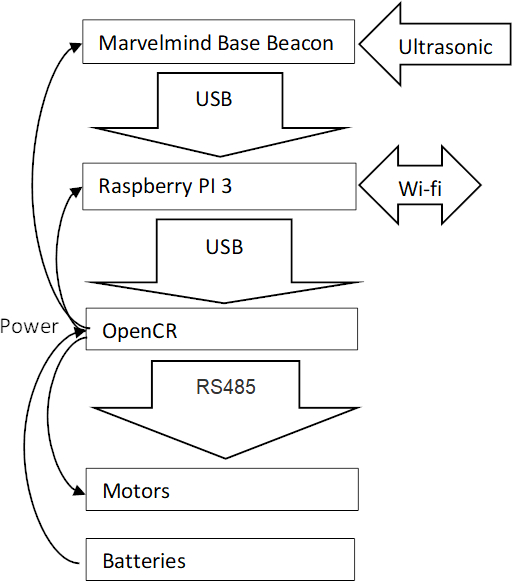
First the user looks at the graphical user interface, which is feeding the user some sort of position of where the ORPS-Robot is in the room and where the desired path for the user is. The user will respond to this information in the system via an Xbox 360 Controller which will be translated into a speed and direction in the basestation. This data will be processed and sent to the M-Transform over Wi-Fi. This published direction and speed from the basestation will be received via the Raspberry Pi 3 on the Turtlebot3 Burger, where it will be fed into a R-Transform on the Raspberry Pi 3. The Raspberry pi 3 will translate this position and speed into something that the OpenCR microcontroller can use for turning the DYNAMIXEL actuator system for movement causing motion of some kind.

Figure 2: The System Architecture

On the ORPS-Robot will exist a Marvelmind mobile beacon which will be sending and receiving data via an ultrasonic signal to four other Marvelmind stationary beacons which will triangulate and calculate the position of the ORPS-Robot via a propagation delay. This data is then sent back to the basestation via Wi-Fi going first through the R-Transform then through the M-Transform where it is processed and presented to the user in the graphical user interface. This closes the loop. It is worth noting that no published ROS commands on the base station or the ORPS-Robot can be sent through the transform itself.

## Hardware Architecture

Figure 3: Overall Hardware Architecture

For this revision of the document we decided to meet the core requirements of the project. We will only talk about the hardware needed to meet the core of this project with the stretch goals to be talked about for a future revision.

While hardware is not the focus of this project, the hardware architecture is an important component and each physical component will be introduced by their power requirements and/or communication protocols. The main hardware components are the Basestation, ORPS-Robot and Marvelmind Beacons. The basestation and ORPS-Robot will communicate via Wi-Fi while Marvelmind stationary beacons communicate amongst themselves via ultrasonic. A mobile Marvelmind beacon will communicate to the Raspberry Pi 3 via USB 2.0. The OpenCR microcontroller will communicate to the DYNAMIXAL Actuator system via serial signal via RS-485.

Figure 4: Hardware Architecture for the ORPS-Robot

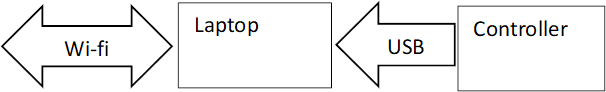
On the ORPS-Robot itself a Li-Po battery feeds the OpenCR microcontroller power which will then feed the Raspberry Pi 3 and Marvelmind mobile beacon. The OpenCR will be feeding power to the DYNAMIXAL Actuator system for movement.

Figure 5: Hardware Architecture for Basestation

The LDS-01 LiDAR will be mentioned briefly but we have not decided whether this is a crucial component of the architecture as it is becoming a stretch goal for the project, as such it is left out of the current set of diagrams.

### Xbox 360 Controller

The human interface device available is a commercial game controller. ROS has a node that can publish the current controller state as a topic. This controller will be connected to the laptop using USB 2.0. The controller joysticks and triggers return an analog value to the laptop while the rest of the buttons work as a digital on and off switch.

### Communication Link

Most communications between devices in this project will be via either USB 2.0 or IEEE 802.11 Wi-Fi apart from Marvelmind which communicates via ultrasonic amongst themselves and USB 2.0 via the base station.

### Marvelmind Indoor Navigation System

Marvelmind indoor navigation system is designed for precise () location data meant for specifically off-the-shelf usage in autonomous robot, vehicle and copters. Locations are calculated via propagation delay of an ultrasonic signal between beacons using trilateration. Stationary beacons will form the map automatically, feeding that data via USB 2.0 to the basestation.

### ORPS-Robot (TurtleBot3 Burger)

The ORPS-Robot is a collection of disparate components meant which combine to make a semi-autonomous robot meant for validating control systems research and to serve as a vessel for undergraduate education. The “Burger” design splits the robot into separate layers where each component is stored. The hardware in the ORPS-Robot is a literal stack with the LDS-01 LiDAR on the top of the robot, Raspberry Pi 3 on the middle stack and OpenCR microcontroller on the bottom stack with the DYNAMIXAL Actuator system existing on the bottom with a single ball bearing for rotational movement.

#### 3.1.4.1. Raspberry Pi 3 Model B

Raspberry Pi is a small computer which we are using as our main controller on the Turtle-Bot. It will be running Linux, and act as our point of contact to the ORPS-Robot. The Raspberry Pi 3 will be connected to the laptop via Wi-fi, and the OpenCR and Marvelmind via USB. The Raspberry Pi 3 will take signals from the laptop as well as the Marvel-Mind sensors, process them, then send them to the OpenCR board. The Pi is equipped with HDMI and USBs which we will use to program it.

#### 3.1.4.2. OpenCR

OpenCR is a controller board used for controlling the different robotic components. This board is open source and made specifically for working with ROS systems. This board is attached to the battery, which it uses to supply power to all the other components on the robot. The OpenCR is attached to the Raspberry PI 3 via USB 2.0 and to the DYNAMIXAL Actuator system via RS-485. The OpenCR is programmable with the Arduino software development environment meaning that in addition to using ROS to control the ORPS-Robot, we have access to addition programming patterns using Arduino’s C/C++ functions and libraries.

#### 3.1.4.3. DYNAMIXAL Actuator system

The motors for the ORPS-Robot use DYNAMIXAL XL430-W250 motors. There is one motor for each of the two wheels. The motors are connected to the OpenCR on a TTL Multidrop Bus using a TTL Half Duplex Asynchronous Serial Communication protocol.

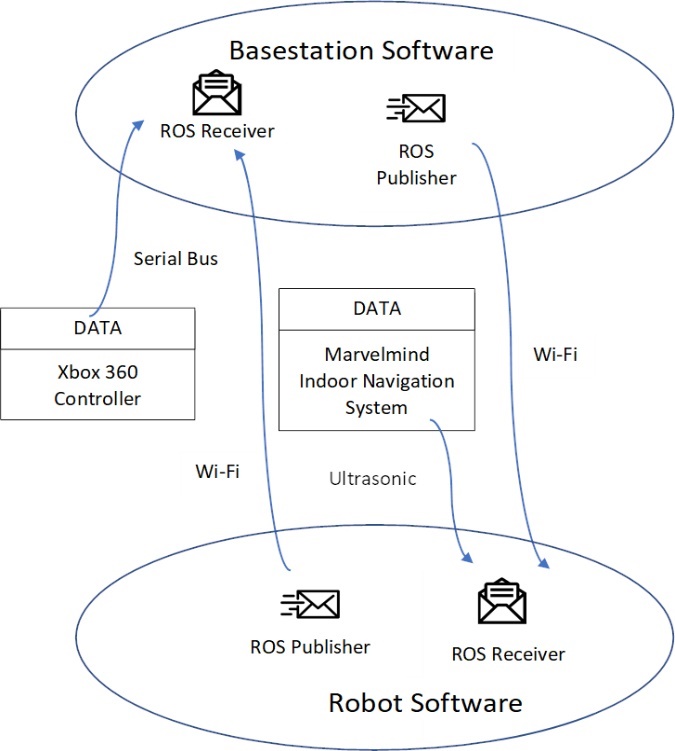
#### 3.1.4.4. Li-Po Battery

The battery on the ORPS-Robot is a Li-Po Battery with a power output of 12V DC, 5A. This battery is connected to the OpenCR board where it is used to power the whole robot.

#### 3.1.4.5. LDS-01 LiDAR

The LDS-01 is a 2D laser scanner capable of sensing that collects a set of data around the ORPS-Robot to use for SLAM (Simultaneous Localization and Mapping) via a USB 2.0 connection on the Raspberry Pi 3. This portion was excluded from the figures as it is not a crucial part of the hardware architecture and may be excluded from the final revision.

## Software Architecture

For this revision of the document we decided to meet the core requirements of the project. We will only talk about the software needed to meet the core of this project with the stretch goals to be talked about for a future revision.

There are two main software stacks, that of the basestation and ORPS-Robot. The basestation software will exist on a laptop, run a graphical user interface for the user controlling the robot to interact and control the robot via an Xbox 360 controller which will also be connected to this basestation. This data will be fed to a ROS receiver and be processed to be set to the ORPS-Robot. When the user makes some change in the controller input that is collected by the ROS receiver and fed through the M-transformation to be received by the Raspberry Pi 3 on the burgerbot via Wi-Fi. We have not worked out what this position data will look like, whether it be an actual video signal via a camera or the positioning data via the LiDAR but there will be some graphical user interface component to this project.

Figure 6: The overall software architecture with ROS publishers/receivers

On the basestation as mentioned previously we will be publishing commands to the burgerbot, including the movement commands via the controller but also doing error measurements between what the user commanded the robot to go and where it went. This is where the path smoothing part of the project comes into play. There will be some desired position and an actual position of the robot. The actual position will be what the Marvelmind ultrasonic indoor navigation system calculates and the desired position will be where the system tells the user to go. The aim is to minimize the error between the actual position and the desired position, even with noisy inputs.

As for the software on the robot itself, it must be capable of receiving data from a Marvelmind sensor which will be placed on the robot itself via USB 2.0. This positioning data will be processed on the Marvelmind itself and will be published for further processing on the basestation. The robot software, which will be running on a Raspberry Pi 3 must be able to receive ROS commands from the basestation and publish the position data to the basestation. As the communication delay is too great to be doing this all on the basestation this means the system must be robust enough to be setup to run then process the information it receives on its own. This also means controlling the OpenCR, a microcontroller, to control the DYNAMIXEL actuator motors on the Burgerbot itself for movement.

### Robot Operating System (ROS)

ROS is a middleware library that creates interfaces for many common robot components and allows them to communicate in standardized ways. Each ROS process is called a node. Nodes communicate through topics or services. A service provides a classic server-client model where the client makes a request and the server responds. A topic implements a logical many-to-many data bus. Nodes can publish to or subscribe from any topic without knowing anything about other nodes on the topic.

### The M-Transformation

The control signals from the human controller will take the form of a 1x2 integer vector indicating speed and direction. As of this writing the actual math has not been explained to us for the M-Transformation other than to say this is a simple linear transformation. Our job is to test this functionality as a black box, not necessarily understand it.

### The R-Transformation

The control signals from the Wi-Fi network will take the form of a 1x2 integer vector indicating speed and direction. As of this writing the actual math has not been explained to us for the R-Transformation other than to say this is more than a simple linear transformation, requiring more operations than the M-Transform. Our job is to test this functionality as a black box, not necessarily understand it.

Figure 7: The McCourt input-output transformations

### The Graphical User Interface

The graphical user interface will be presenting the user with some sort of positioning of where the robot is in space. Whether it be through the LiDAR or webcam, this has not been decided as of this writing of the specification. There will also be some sort of path we desire the user to follow presented onto the screen.

# System Design

## Hardware Design

Our hardware is largely preset

### Objectives

### Constraints

* Raspberry Pi 3 (32-bit ARM Cortex-M7)
* OpenCR (32-bit ARM Cortex-M7)
* DYNAMIXEL Actuator system

### Composition

### Uses and Interactions

### Interface

### Resources

### Details

## Software Design

### Objectives

### Constraints

### Composition

### Uses and Interactions

### Interface

### Resources

### Details

## Human Interface Design

### Functionality from the User’s Perspective

When in remote control mode the robot can be controlled by a consumer game controller. Ideally the user will be presented with a graphic of a 2-dimensional field with a marker representing the robot’s current position and another at the desired position. The user will be tasked with attempting to bring the robot to the desired position as it moves.

The task will be repeated with and without the McCourt filter in line.

### Interface Objectives

### Interface Constraints

### Use Cases

# Ethical Considerations

In this project, our specific goals do not raise many ethical concerns, however, there are still things that could happen from this project that should be considered.

Future use of McCourt’s filter: The primary purpose of McCourt’s algorithm is to help with the delay of human interacting in closed loop control systems. This algorithm could be used for military systems or other weapons systems. This is not directly our concern as McCourt’s algorithms are not ours, but our work in enabling it could be considered helping. This concern is mainly dependent on the motives and intensions of the IP owner. With our countries current IP laws and our confidence in the owner of the IP we are not in any way currently worried about this ethical concern.

Complete testing: McCourt’s filter could be use in the future in systems that could cause harm if they were to malfunction. Because our testing is a proof of concept, it is partly our responsibility to make sure this algorithm works in practice, so there is little chance of malfunctions in the future of this algorithms life.

Use of funds: This project is funded by Dr. McCourt who has funding from the University of Washington. We did not technically earn this money, so it is our responsibility to use this money efficiently, and to not cause money to be spent due to mistakes.

# References

1. Turtlebot3 Manual: <http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/>
2. Robot Operating System: <http://www.ros.org/>
3. Open Source Computer Vision Library (OpenCV): <https://opencv.org/>

# Errata

None Currently