Optimal Control of Image Based Visual Servoing (IBVS) for High Precision Visual Inspection Applications

Mark S. Allen
Clemson University, allenmark.1994@gmail.com

OPTIMAL CONTROL OF IMAGE BASED VISUAL SERVOING (IBVS) FOR HIGH PRECISION VISUAL INSPECTION APPLICATIONS

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Mechanical Engineering

---

by
Mark Stewart Allen
December 2020

Accepted by:
Dr. Laine Mears, Committee Chair
Dr. Bryan Riley
Dr. Oliver Myers

ABSTRACT

Visual servoing is a control technique that uses image data as feedback in a motion control loop. This technique is useful in tasks that require robots or other automated motion systems to automatically inspect parts or structures in motion. One specific method of visual servoing is Image Based Visual Servoing (IBVS), a method that simply minimizes the differences between an observed image orientation and a desired one. This method works well for orientations where the differences are small, but in the case where the desired orientation is more difficult to reach, the system can become unstable, either driving to infinity through a phenomenon known as camera retreat or following non-optimal and non-repeatable trajectories. This work attempts to address camera retreat and other non-optimal paths by applying dynamic programming, an optimal control method that can determine an optimal trajectory by partitioning possible trajectories into multiple smaller trajectories. Using a cost function to penalize undesirable sub trajectories, the optimal overall trajectory can be determined and initiated. This work attempts to explore an optimized portioned approach using dynamic programming to address camera retreat. The motivation for this is to create a high precision visual servoing sequence suitable for high tolerance automated processes; specifically, quality inspection of airplane wire harnesses.

DEDICATION

I would like to dedicate this thesis to my family and friends who have supported me up to this point. I would also like to dedicate this to Mrs. Serita and the PEER/WISE family who have given me a home throughout my undergraduate and into my graduate studies. I would also like to dedicate this to specific people who have helped me along my research journey: Dr. Freddy Paige, a mentor to my research and life as an academic since undergrad, Shyla Wright, a friend with a technical background outside mine for giving me an outside perspective, Sanaa Lucas for having writing sessions with me during every stage of writing my thesis, Thelmon Harris for giving me encouraging words throughout this writing process, Alexander McIntosh for also giving encouraging words throughout this writing process from Germany. Lastly, I would like to dedicate this to all of my mentees who have motivated me to push for my goals more than they know. I hope that this work inspires you as much as you all have helped and inspired me.

ACKNOWLEDGMENTS

I would like to acknowledge SAFRAN for funding this project that lead to my research questions. I would also like to acknowledge my advisor, Dr. Laine Mears for providing the resources for building various prototype systems, books used for learning about automated visual systems, and support for presenting my work at conferences. I would also like to acknowledge members in my lab for helping with the physical assembly of the rather large prototype system developed in this work. Lastly, I would like to acknowledge Ethan Wescoat for the long hours developing the initial mathematical framework that laid the framework for this thesis. This work would not have been possible without you, and I thank you for this aid.

TABLE OF CONTENTS

Table of Contents (Continued) Page

LIST OF TABLES

LIST OF FIGURES

List of Figures (Continued) Page

CHAPTER ONE

Introduction

Visual servoing is an image-based control technique that involves collecting image data from a vision sensor (typically a camera) as an input to a control loop. This input is used to determine how to control and drive a system. Vision-based robotic systems using such techniques are employed in a variety of manufacturing industries for inspection within the food, medical, aerospace, and automotive industries. A major advantage of vision-based robotic systems is systematically avoiding the introduction of potential error by humans, who may become apathetic and miss details of the inspection criteria [1]. With the rise of these systems in so many areas, research in visual servoing is becoming more critical so these systems can continue to meet the demands of inspection in manufacturing, specifically to improve the accuracy, optimality and robustness of vision based control systems [1][2][3][4]. Reducing the cost of these systems is also important and considered a goal.

Image Based Visual Servoing (IBVS) is a common control scheme used in automated visual inspection processes. The simplicity of implementation as well as the ability to always drive the system to the desired solution makes the method adequate for most applications [5][6][7][8]. However, in the traditional control scheme a phenomenon known as *camera retreat* can often drive the system to instability or cause the system to follow a non-optimal trajectory while trying to reach the solution. Specifically, a singularity in the calculation causes the system to approach infinity. This limits the possible implementations for tasks that require precision inspection or tasks that have physical constraints that limit the motion of the system. The details on the framework of this control scheme and the limitations of camera retreat will be discussed in detail. When using IBVS, inspection must also be executed at a speed that is timely, cost effective and does not set back production [1]. The quality of the image being used as an input must also be high enough to correctly interpret the image for the controller. This ensures that the system drives itself to the correct state based on the input. The ability to capture high-quality images opens the possibility for high precision inspection applications, and further emphasizes the need for optimal control of IBVS to capture usable visual inputs [9]. One way to ensure high quality image capturing and precise actuation is by implementing optimization techniques into common visual servoing control schemes. This combination of optimization and control is known as optimal control theory [10][11].

Optimal control theory is the practice of determining a control law for a system to reach a certain set of optimality criteria. For a robotic system, common optimization criteria include the minimization or maximization of time, efficient path planning, and optimal trajectories for driving a system to a desired state. The mechanical constraints of the system like motor torque limitations, battery life, or physical limitations of the robotic system allow one to determine if and how an optimality criterion can be satisfied based on the specifics of a system. To ensure the system's motion is optimal and to combat camera retreat, the intention of this work is to introduce an optimal control method known as dynamic programming to the traditional visual control scheme.

Dynamic programming is a recursive optimization technique of selecting the optimal path in a partitioned space by calculating the minimum *cost to go* using a specified cost function. The cost function is determined based on system constraints. Starting from the desired state of the system, the minimum cost to go for each state in the partitioned space is calculated recursively until the current state is reached [11][12]. This process exhausts all possible trajectories, allowing the optimal state sequence to be determined which in turn determines the optimal state trajectory. The details of this approach will be further discussed in Chapter 2.

We can combine dynamic programming with IBVS through the image Jacobian matrix (or interaction matrix), which is a matrix representation of all the intrinsic camera parameters. The interaction matrix characterizes the image dynamics and can be directly incorporated into the cost function of a dynamic programming control scheme. This allows the minimum *cost to go* to be calculated for a system based on the image dynamics of the system, creating an optimal IBVS control method that can handle the *camera retreat* phenomenon as well as trajectories that are not optimal.

In Chapter 2, an introduction to visual inspection, dynamic programming, and IBVS is presented. In Chapter 3, the framework to address a specific scenario of IBVS in which the system follows non-optimal trajectories is developed. Chapter 4 gives an example application, while Chapter 5 describes suggestions for future research.

CHAPTER TWO

Background and Literature Review

2.1. Visual Inspection

Visual inspection is a common step in manufacturing processes to determine part quality. It also is a key process in a variety of other industries, for example construction where one may need to verify the structural integrity of a building [4][9][13][14][15][16]. Inspectors look for both cosmetic and functional defects in the production process, or in the case of structural integrity, evaluating cracks and deformities in buildings that could cause the structure to collapse [1]. With inspection becoming more integral in so many industries, the push to accurately automate this traditionally manual process is becoming more critical to save cost and time. Performing manual inspection requires a physical person to complete the task which opens the door for human error. These sort of tasks also tends to be viewed as tedious which can cause inspectors to conduct the inspection inadequately [17]. Automating this process eliminates the need for human interaction and in most cases, performs the inspection in a more accurate and effective way. Automation however, is still not a trivial process and does not automatically ensure better performance. It depends heavily on the systems used to develop the processes [18]. To automate a manual inspection process, the tasks must be broken into three stages: visual input, image processing and output inspection [5]. During the input stage, the images are captured, loaded into the system, and stored. In the processing stage, the predetermined features of interest are extracted and compared to validate correctness with respect to desired colors, orientation, and other features desired by the user. The output of the inspection is what the user sees to perform further analysis and verification. Processing the output can also be automated depending on the application. The contributions of this work are aimed at the input stage; specifically, improving the ability of a system to accurately and optimally position itself to capture the best image for inspection. In order to make sure the best picture is taken; robust and optimal image-based control strategies are necessary so servo the system.

Methods for visual inspection have become over time more advanced and efficient, taking advantage of accelerating camera and image enhancement technologies to better determine and classify features of interest. In a continuance of that trend, inspectors have also placed cameras on automated systems such as mobile ground robots or unmanned aerial vehicles in order to be more efficient or to reach areas beyond the ability of humans [19]. Along an assembly line, cameras placed within production can gather data on each part [20], while unmanned drones taking pictures for building inspectors can help with areas that are unsafe or inaccessible; these systems also free up human capacity. However, image analysis is still key to determine and classify faults, in order to more fully automate the inspection process. To that end, computer models of image recognition need to be trained through machine learning [21].

Automated Visual Inspection (AVI) is a class of image processing methods for quality control which covers a wide range of different techniques based on filtering, learning, and a hybridization of different methods [17]. Precision placement of cameras or image capture devices is critical to such methods' success; visual servoing is often used to actively adjust position based on application [22][23]. This means that by improving the precision and accuracy of visual servoing techniques used in AVI processes, we get higher quality images and more accurate inspection. Visual servoing allows for images to be taken from different angles and magnifications, in order to classify different types of faults. However, the introduction of this motion introduces complication in analysis due to the need for the image processing algorithm(s) to understand the correct orientation and part position. In addition, the camera should minimize the travel time to the target pose in order to maximize the number and quality of images [5]. While so many applications have benefited from the advances in computer vision, high precision applications still require more precision in the visual control methods used to be considered the standard. For example, in manufacturing environments where the objects of interested are in complex configurations and difficult to reach or with larger parts that require multiple images to be captured for inspection. While these problems present themselves in a wide variety of manufacturing environments, we have seen these problems frequently in the aerospace industry as they look to move to more automated manufacturing processes.

The aerospace industry is known for its tight tolerance and heavily manual processes, however with the advances happening in automation, this industry is also looking to automate; specifically, automated inspection for pre and post analysis of parts for quality assurance. Our lab partnered with the company, SAFRAN, a leader in the aerospace manufacturing industry and specifically, airplane wire harness manufacturing, to explore ways to automate some of their inspection process. Airplane wire harnesses are complex and are manufactured in a wide variety of configurations that require precise inspection to differentiate between wire harness features and connection points. Incorrect assembly in these connections points lead to catastrophic failure in operations. This precision in the manual inspection meant that the automated process had to be optimal and precise if it were to be implemented. After exploring what image control techniques were commonly used in manufacturing environments, Image Based Visual Servoing (IBVS) appeared to be the standard due to its ease of implementation and ability to always converge on the desired camera pose. In chapter three, we will explore in detail how some of the early attempts to help automate the inspection of a process that has tight tolerances helped to motivate the investigation into more precise methods of image-based control. In the next section, IBVS will be explained at a high level as well as the mathematical framework for how it works and its shortcomings.

2.2. Image based visual servoing

Image Based Visual Servoing (IBVS) is a method of visual servoing in which the control is based on dynamics from the image points located in the image plane. In this section we will give the necessary background information of the IBVS frame work using nomenclature that is used in the literature. As mentioned earlier, IBVS is used in a wide variety of control schemes to servo and control various types of robotic systems [5][24][25][26]. The basic principle behind IBVS is to simply minimize the difference between a desired image orientation and the image orientation being displayed on the camera. Figure 2.1 shows a visual representation of this idea. This simplicity is what first motivated the use of IBVS for quality inspection. Desired features of the parts were already known and the control law, in theory, could easily servo to specific features on the wire harness.

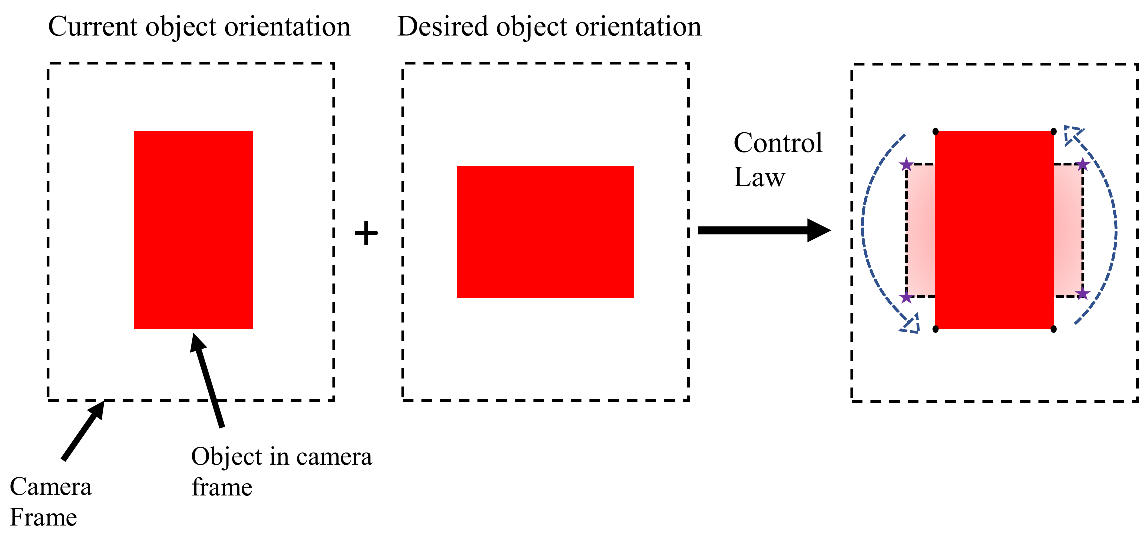


Figure 2.1: Visual representation of the basic idea behind image based visual servoing. The desired object orientation is known, and the goal is to minimize the differences between known and current image orientation.

While conceptually IBVS is simple to understand, there are multiple aspects of an automated system that must be considered before implementing. Limitations in the environment, computational power, and mechanical constraints are important entities to consider in the control scheme. Figure 2.2 is a well known control scheme that most robotic systems implementing IBVS use [27].

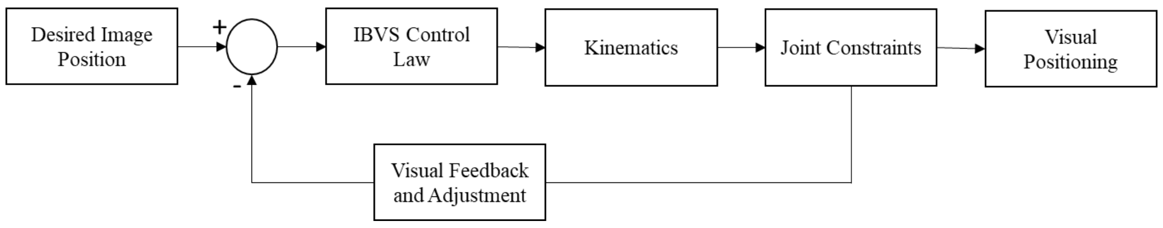


Figure 2.2: Block diagram representation of IBVS control scheme in robotic control scheme. This diagram was used for initial testing of IBVS once we implemented it on our system.

The kinematics (also referred to as the equations of motion) include the limitations of the geometric representation of the robot. While we must consider these limitations when assessing what the robot can accomplish, we must also consider the physical capabilities of the motors or joints of the robot. In theory, the IBVS control law can then drive the system closer to the desired image position considering these constraints. It will be shown later that IBVS does not handle these constraints well. The control law used in this block diagram is derived from the image dynamics and is outlined below. Let Equations (1) and (2)

$$r = (x, y, z)^T \tag{1}$$

$$\dot{r} = \left(T_x, T_y, T_z, \omega_x, \omega_y, \omega_z\right)^T \tag{2}$$

represent the coordinates of a robot end-effector and its velocity, respectively in a 3 dimensional space [28]. The vector $\dot{r}$ is composed of the linear and angular velocities, which can be separated into Equations (3) and (4), which are

$$E = \left(E_{x2}, E_y, E_z\right)^T \tag{3}$$

$$\omega = \left(\omega_x, \omega_y, \omega_z\right)^T \tag{4}$$

where T is the linear velocity and ω is the angular velocity [28]. Let Equation (5)

$$f = (u, v)^T \tag{5}$$

be the image-plane coordinates of a point in the image and Equation (6)

$$\dot{f} = (\dot{u}, \dot{v})^T \tag{6}$$

be the corresponding velocity [28]. In order to form the control law, we must also include what is known as the interaction matrix, which relates the motion of the camera to the motion of the feature point. The derivation of Equation (7) can be found in Appendix A.

$$L_s = \begin{bmatrix} \frac{\lambda}{z} & 0 & -\frac{u}{z} & -\frac{uv}{\lambda} & \frac{(\lambda^2+u^2)}{\lambda} & -v \\ 0 & \frac{\lambda}{z} & -\frac{v}{z} & \frac{-\lambda^2-v^2}{\lambda} & \frac{uv}{\lambda} & u \end{bmatrix} \tag{7}$$

Here λ is the focal length of the camera. We can combine these camera properties to get the velocity of the image coordinates with respect to the camera properties by multiplying the interaction matrix by Equation (3), resulting in Equation (8).

$$\dot{f} = L_s(f, r)\dot{r} \tag{8}$$

This equation can then be used to form a control law based on the physical limitations of the camera in combination with the with the image space dynamics. A simple and widely used control for this type of control scheme is

$$U = kL_s^{-1}(f,r)\dot{f} \tag{9}$$

where k is a gain matrix and U is the control input.

While the control law for IBVS is relatively simple, there are two main problems that arise with implementation, the first being constraint handling. The inability of IBVS to handle various constraints including robot workspace constraints, visual constraints, and limits on the robotic system itself motivated the work of a variety of research in constrained IBVS. This is typically done by formulating the problem as a nonlinear problem. One approach that has been successful is the work done in [12]. Here, they developed a method of combining Model Predictive Control (MPC) with IBVS, formulating what they call Visual Predictive Control (VPC). MPC works well due to its ability to constrain the inputs and outputs of the system directly into the control law. In the case of IBVS, the constraints are limitations of the image coordinates of the desired image. These limitations are driven by the physical constraints mentioned earlier. While VPC aided in helping IBVS handle constraints, it still fails in addressing the second shortcoming of IBVS: following nonoptimal trajectories. In certain cases where depth is of a certain value relative to the image coordinates, the system follows nonoptimal trajectories when converging on the desired camera pose. One nonoptimal trajectory that occurs in most applications is called camera retreat. In this scenario, the control causes the system to converge to infinity and never reaches the desired posse. IBVS has previously been subject to attempts to apply optimal control to a prescribed path addressing camera retreat using partitioned approaches, however most partitioned approaches are not optimal. This motivates the use of investigating dynamic programming (DP), an optimization technique that can be partitioned. While MPC will not be used explicitly, the derivation for optimizing IBVS using DP follows a similar pattern due to the nature of most control optimization techniques. The details of the derivation will be discussed later, however first the problem of camera retreat will be explained.

2.3. Nonoptimal trajectories and camera retreat

The two main problems with the IBVS control scheme this work aims to address are the emergence of nonoptimal trajectories when converging on the desired camera pose as well as the camera retreat phenomenon, which causes the system to drive to infinity while attempting to converge [29][30][31]. The beauty of IBVS is that the system will always try to converge to the desired camera pose and will never approach a camera pose that it is not wanted, however the path the system takes is not always optimal, sometimes even in trivial cases. The issue stems from depth estimation of each of the feature points of desired camera pose. The interaction matrix requires knowledge of depth in order to accurately compute the interaction, and this depth does not always remain constant in practice nor is it always known and attainable. The techniques used to address this problem often include a sensor typically stereo techniques involving multiple cameras, to estimate depth in real time [28]. These techniques do aid in smoothing out the paths to looking more “normal”, however these methods are not always practical depending on the application and the path to convergence may still not be optimal. For the applications in this thesis, we assume depth to be constant, however even at constant depth, the phenomenon of camera retreat still is an issue; often worsening depending on the depth and the degree of rotation in the *z* axis. Typically, camera retreat happens with larger rotation, however the phenomenon can happen with small rotations as well. To illustrate the sensitivity of camera retreat to certain rotations, a simulation where the object of interest is a square is shown in Figure 2.3 and Figure 2.4. Each simulation has two different initial offsets in the *z* axis with the same goal camera pose. In these simulations, the square where the vertices are stars indicate the desired camera pose, and the square with circle vertices indicates the initial camera pose. The solid blue lines represent the path the square took to reach the desired pose. Figure 2.3 illustrates how the control law drives the system to the desired camera pose following the obvious solution of traveling in a straight line to reach the desired camera pose. Figure 2.4 illustrates the problem of camera retreat where it converges to the desired camera pose after first retreating far away in the *z* axis. Despite the goal camera pose remaining the same, in Figure 2.4 we see the slight change in rotation causes the camera to retreat to infinity. IBVS struggles in handling rotations due to singularities that occur in the interaction matrix. These singularities can occur at any point in the control algorithm and are mainly addressed as they occur in a specific application. Both simulations we run at the same initial depth.

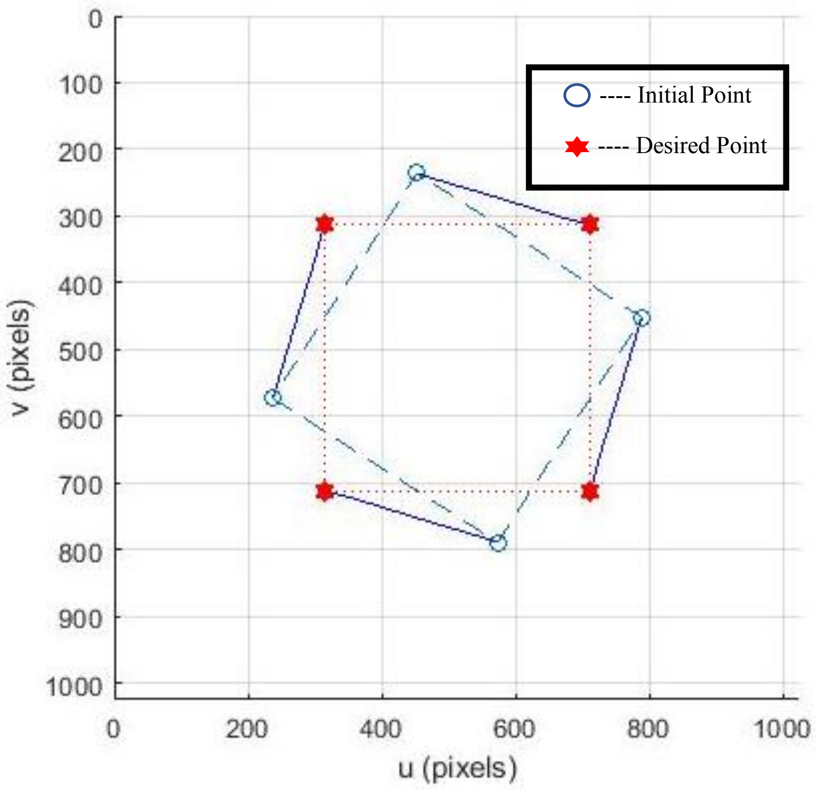


Figure 2.3: Case of IBVS converging optimally with respect to time to a known camera pose. certain rotations about the *z* axis, IBVS converges in an intuitive way.

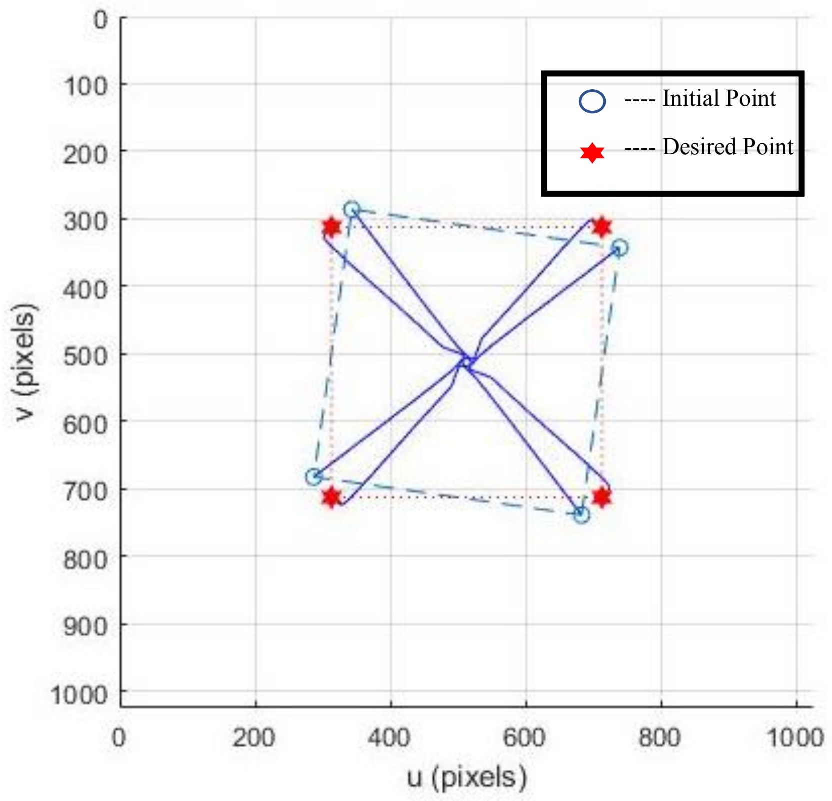


Figure 2.4: Case of IBVS with same goal camera pose at different initial orientation around the *z* axis. The slight change in rotations causes a drastically different path to the desired camera pose despite the solution being trivial.

The details of this simulations will be discussed in a later section; however, it is important to note that these simulations were run with the unconstrained IBVS control law outlined above.

`When first examining the problems of camera retreat, it may seem like there may not be applications where large rotations would be a concern, however what makes the issue worth examining is the fact that small rotations can also trigger camera retreat depending on the depth. Small rotations can stem from vibrations on the system or even occur in the process of trying to reach a desired pose along the trajectory. As mentioned before, there are other partitioned approaches to solving this issue, however non guarantee optimality. Dynamic programming is one partitioned technique that partitioned the environment to penalize following non optimal trajectories while also providing an optimal solution. In the next section, dynamic programming (DP) will be examined, along with the ways in which it could address the shortcomings of IBVS.

2.4. Dynamic programming

Dynamic programming is a method of optimal computation involving decomposing a problem into smaller subproblems. The smaller subproblems act as steps for the overall problem solution; the solutions to these smaller subproblems are obtained and then indexed to determine the optimal solution, meaning it works best for optimizing solutions that are recursive in nature [11][32][33][34][35][36][34][37]. To better understand this, it is necessary to first understand the Bellman principle of optimality:

> ***An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision*** [11].

This principle is the underlying foundation as to why dynamic programming works. Once a path is deemed optimal to reach the solution, all subsequent paths are also deemed to be optimal. To further illustrate this example, consider the path below in Figure 2.5 going from point *A* to point *C*.

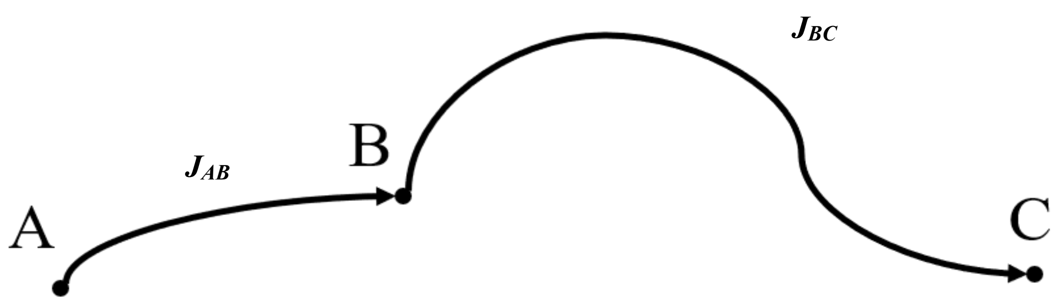


Figure 2.5: Optimal path from A to C with associated costs. J_{AB} represents the cost go from point A to B and J_{BC} represents the cost to go from B to C

To prove the principle of optimality, we first assume the path in Figure 2.5 is the optimal path from point A to C. According to the Bellman principle of optimality, this means the path from B to C is also optimal. We measure this in the form of a cost value, which is typically a value determined by a function of interest that is being minimized. This function could be functions of time, displacement, or even something like battery usage. In this example, we use arbitrary costs J_{AB} and J_{BC}, which are the cost to go of an arbitrary performance measure to be minimized from point A to B and from point B to C, respectively. The cost to go from A to C, which is also the optimal path in this example, is given by J^*_{AC}. The optimal cost from A to C can be represented by Equation (10) as

$$J^*_{AC} = J_{AB} + J_{BC} \tag{10}$$

The "*" in this notation indicates that this is the optimal path from A to C as there are more nonoptimal paths from A to C. Let us consider one of these alternative paths to show, by contradiction, that if in fact J_{AC}^* is the optimal, J_{BC} must also be optimal from *B* to *C* based on the Bellman principle. In Figure 2.6 below, another potential path from *B* to *C* is shown with and associated cost.

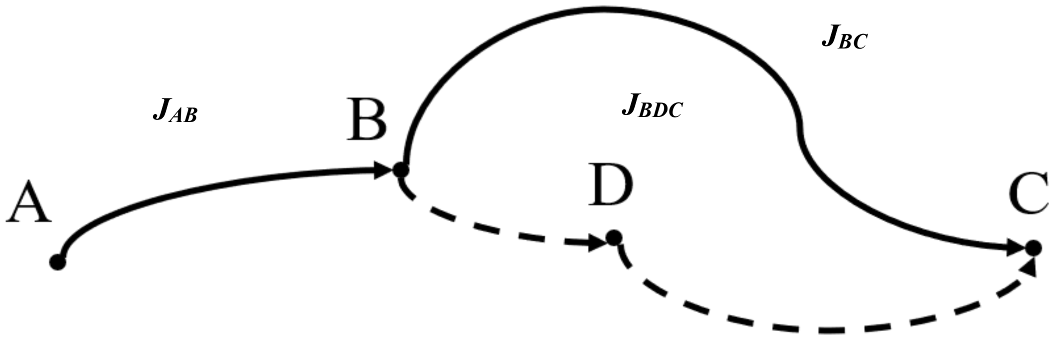


Figure 2.6: Optimal path from *A* to *C* with two potential paths from *B* to *C*.

Here the cost from *B* to *D* to *C* is given by J_{BDC}. To prove by contradiction, we first assume the new path *BDC* is the optimal path from *B* to *C*, which means that

$$J_{BDC} < J_{BC} \tag{11}$$

In terms of the optimal cost to go from point *A* to *B*, this means that the following equation must also be true

$$J_{AB} + J_{BDC} < J_{AB} + J_{BC} = J_{AC}^* \tag{12}$$

The only way Equation (12) can be true is by violating the initial assertion that the optimal path is *A*-*B*-*C*, thus by contradiction, *B*-*C* must also be the optimal path from *B* to *C*. It can be shown how this can be extended to a multistage decision-making process by considering Figure 2.7.

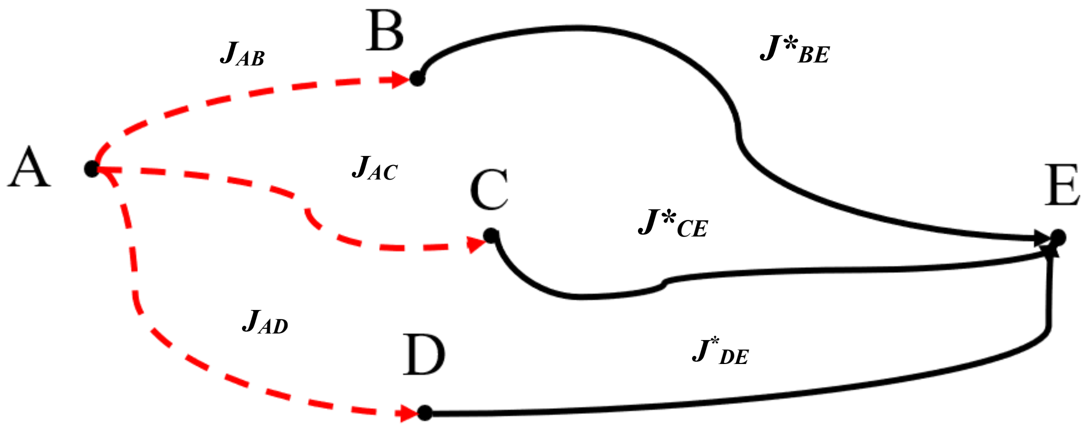


Figure 2.7: Multi-stage decision making using principle of optimality. Optimal paths from *B* to *E*, *C* to *E*, and *D* to *E* are known.

Let us consider a system going from state *A* to state *E* following the most optimal trajectory. Currently, the optimal path for *B-E*, *C-E*, and *D-E* is known along with the associated cost $J^*_{BE}, J^*_{CE}, J^*_{DE}$. What it is not known and is if path *A-B*, *A-C*, or *A-D* is the optimal path to take from State A. This is the subproblem dynamic programming aims to compute and solve. If it is determined that, for example, *C* is the next state the system can achieve optimally, then by the Bellman principle, *C-E* is the optimal, terminal path to take and path *A-C-E* is the optimal path from state *A* to *E*. The same applies for path *A-B* and path *A-D*. In order to find which of the three paths is optimal, the total cost for each potential path must be calculated and compared via Equations (13) (a-c).

$$J^*_{ABE} = J_{AB} + J^*_{BE} \tag{13.a}$$

$$J^*_{ACE} = J_{AC} + J^*_{CE} \tag{13.b}$$

$$J^*_{ADE} = J_{AD} + J^*_{DE} \tag{13.c}$$

The optimal path can be determined by calculating the minimum cost of these equations. Dynamic programming extends this decision-making process to a sequence of decision- to determine the optimal trajectory and path. The next section examines how this computational technique can be combined with IBVS to both optimize and constrain IBVS.

2.5. Dynamic programming and IBVS

In this section we formulate the IBVS optimal control problem with dynamic programming. As mentioned previously, the goal of dynamic programming is to determine the optimal trajectory by calculating smaller sub trajectories through a cost function [38]; for the vision positioning application, the cost function represents the error e between a desired point and the current pose of the camera [27]. Equation (14) shows this error term as a vector of feature points of the image.

$$e = s_d - s \tag{14}$$

Here, s_d is the desired point and s is the current pose of the camera. Discretizing this equation allows it to be implemented into a stepwise algorithm to determine the error at each time step. The discretized version of this equation is given in Equation (15):

$$e(k) = s_d(k) - s(k),\ k = 1,2,\ldots..n \tag{15}$$

The variable n is the number of time steps. There are three main constraints that are considered: mechanical constraints, which are based off the geometric model of the automation system (and also determine the camera pose), visibility constraints that ensure that the visual measurements stay in the image plane, and control input constraints which are based on actuator limitations [27]. The geometric model is represented by *p(k)* and is dependent on individual joint measurements *q(k)*, as represented by Equation (16) and (17) below.

$$p_{min} \le p(k) \le p_{max} \tag{16}$$

$$q_{min} \le q(k) \le q_{max.} \tag{17}$$

The visibility constraints are based on the predictive model of the trajectory s_m and given by the following relationship:

$$s_{m_{min}} \le s_m(k) \le s_{m_{max}} \tag{18}$$

Lastly, the control constraints can be represented by Equation (19):

$$U_{min} \le U(k) \le U_{max} \tag{19}$$

where *U* is the control input. A cost function that is commonly used for minimizing IBVS is given by Equation (20)

$$J(U) = \sum_{j=k+1}^{k+N_p} [s_d(j) - s_m(j)]^T Q(j) [s_d(j) - s_m(j)] \tag{20}$$

Where *Q(j)* is a symmetric definite-positive identity matrix. N_p represents the horizon over which the cost function is to be optimized. For dynamic programming, the larger this horizon, the more optimal the solution. Using this cost function in simulation, the optimal control sequence can be found with the goal of minimizing the control input.

To calculate the model s_m, the following equations are used as an approximation of Equation (8):

$$s_m(k+1) = s_m + s'_m \tag{21}$$

$$s'_m(k) = L_s(k) T_e \dot{r}(k) \tag{22}$$

$$s_m(k+1) = s_m + L_s(k) T_e \dot{r}(k). \tag{23}$$

L_s is the interaction matrix given earlier as Equation (7), T_e is the sampling period between states, and $\dot{r}$ is the camera velocity.

To final step before implementing dynamic programming is to penalize the control input. This is to ensure that our system does not reach an optimal solution through a control that is not achievable. In this case, the camera velocity is the control input. From this, Equation (24) is derived as

$$J(U) = \sum_{j=k+1}^{k+N_p}[s_m(j) - s_d(j)]^T Q(j)[s_m(j) - s_d(j)] \; + \dot{r}(k)^T * R * \dot{r}(k) \tag{24}$$

R is a weighted matrix for the control variables. To calculate the control term $\dot{r}(k)$, the following equation is used:

$$\dot{r}(k) = (s_m(k) - s_m(k+1))/(\,L_s(k)T_e) \tag{25}$$

Using Equation (25), the optimal path can be determined with respect to the velocity of the overall system. The path is constrained with respect to the velocity upper and lower bounds

$$-.1 \;\leq \dot{r}(k) \;\leq\; .1 \tag{26}$$

and the minimum cost is determined by Equation (24) for each control input. The goal of this cost function is to minimize the time it takes to reach the desired camera pose. This will in turn, minimize path traveled to reach the desired pose making it optimal.

As mentioned earlier, the goal is to focus on a specific set of cases and not to redefine the entire IBVS control scheme. We will see there are cases of IBVS in the next chapter where the trajectory is time optimal and similar to the combined DP and IBVS control scheme.

2.6. Summary

In this section we have reviewed some of the common methodologies in vison-based control and visual servoing. With automation becoming more and more popular in a variety of different industries, the need for robust and optimal visual control techniques is becoming more necessary. In the aerospace industry specifically, this precision will be critical in the implementation of more automated systems as the tolerances and requirements are tight specific. Image Based Visual Servoing (IBVS) is a common visual servoing technique used in the manufacturing industry, however there are shortcomings like camera retreat and following nonoptimal trajectories that prevent it from being used in scenarios where precision is key. These shortcomings stem from the difficulty in estimating depth, a key component in calculating the interaction matrix in the traditional control scheme. While there have been attempts to more accurately estimate depth, these methods often involve drastic modifications to the system and are not always practical. This motivates the need for partitioned approaches to IBVS to deal with camera retreat and non-optimal trajectories. While these attempts have been useful in converging the system in a way that is not completely abnormal, the trajectories are not always optimal, again limiting the application. Dynamic Programming (DP) is an optimal control technique that both partitions the IBVS control scheme while ensuring optimality.

In Chapter 3 we will expand upon the combined formulation above and see specific examples of combining DP with IBVS and how it affects the performance. MATLAB simulation will be used to compare the results of the traditional IBVS control scheme as well as the combined IBVS and DP control scheme. In order understand how this technique measures up to less eloquent visual control techniques, we will also look at a case study done in the early stages of attempting to automate inspection of the airplane wire harnesses. These attempts were the main motivation into attempting to implement IBVS as well as a good baseline comparison to evaluate performance.

CHAPTER THREE

Case Study and Experimental Results

In this chapter we will show the development and implementation of a system used to conduct a case study on feature detection. The case study shows how critical vision-based control techniques are for applications that require high precision. The chapter will be outlined as follows: introduction of the problem of the case study, difficulties in detecting features on wire harnesses, prototype development of system used for case study, image processing and capturing techniques used for feature detections, and lastly, problems with synchronization of motor control and image processing.

3.1. Case Study Introduction

SAFRAN is one of the leading manufacturers of airplane wire harnesses in the world. As a leader in this industry, working with them gave a unique perspective on the future of automation of manufacturing processes in the aerospace industry. Assembly and inspection in this industry are typically a manual process due to the tight tolerance requirements. A mistake in any stage of this process, whether it is improper inspection, improper assembly, or improper placement in the airplane could lead to catastrophic failure during operation [39][40][41]. This typically discourages efforts in automating any stages of the process of assembly and inspection due to most automated process not having the robustness to perform the operation as precise and accurately as a human. Because of these concerns, there has never been a serious attempt to automate by SAFRAN until now. SAFRAN challenged us to perform a case study to assess the potential of automating the inspection process of airplane wire harnesses in hopes of expanding what was discovered to other stages of the assembly process. In the next section, some of the issues that come with attempting to automate the inspection process are explained as well as the complexities that come with inspection of airplane wire harnesses.

3.2. Airplane wire harnesses

Airplane wire harnesses were first developed in order to aid in routing the connections for various electrical components on an airplane. As airplanes developed over time, the number of wires necessary for routing increased to the point where it became a lengthy process when setting up the electrical components of an airplane [42]. Eventually the idea came to assemble the wire harness as they would need to be routed inside the airplane. For example, if the connections for the electrical wiring for the lighting of a specific section of the aircraft was near the connections for say, the power of the HUD display in the cockpit, those wiring connections would be grouped together during the assembly process. From Figure 3.1 below, we can see how airplane wire harnesses can be complex in wire routing and physical dimensions depending on the harness specifications. Some wire harnesses can stretch upward of 30 meters on what is known as a form board (FB) with extremely tight orientation requirements at every section. This can cause inspection to be critical as one misplacement could mean the airplane fails at a critical point. It is not uncommon for the wire harness to be inspected multiple times throughout the assembly process and a mistake to still be discovered despite all these precautions to ensure correct assembly. These frequency in inspection motivated the idea to automate this process, with the idea being that while the workers assemble the wire harness, an automated inspection system could inspect the harness as the same time.

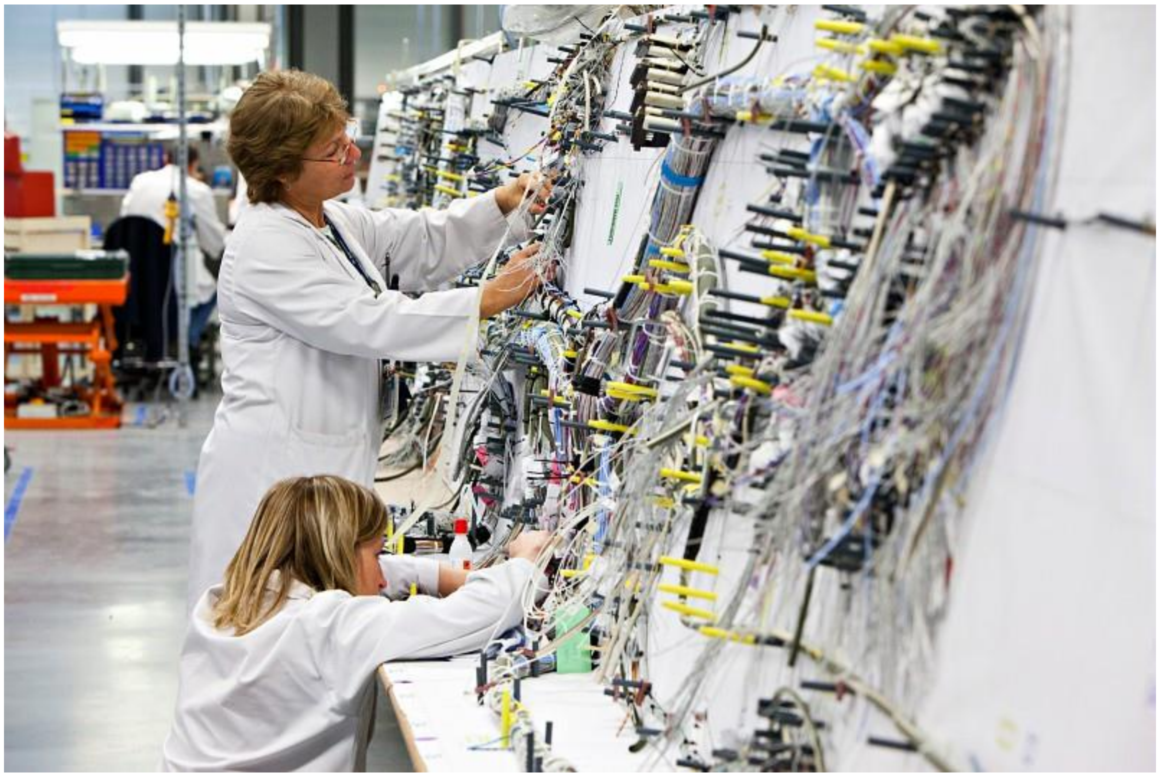


Figure 3.1: SAFRAN team members assembling airplane wire harness assemblies for a military aircraft. The complex shape and number of critical components make this a heavily manual process. There are many opportunities for mistakes during assembly.

Having a system inspecting this way would greatly reduce the need for manual inspection and would save time and money. Workers could put their efforts in other aspects of the assembly process and increase production rates.

Given the length of some of the wire harnesses, the layout of the facility, and the requirement that the system could not interfere with the work of the workers, a track type system where inspection system would be attached became the leading solution to the problem. Figure 3.2 shows the proof of concept of the system in a manufacturing

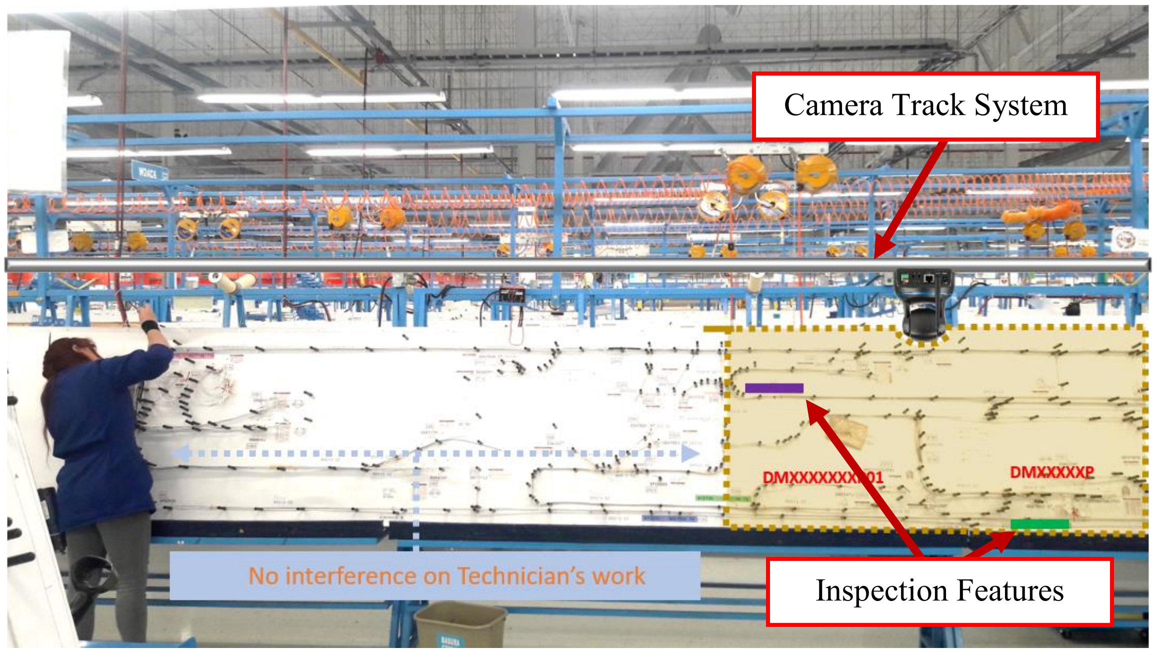
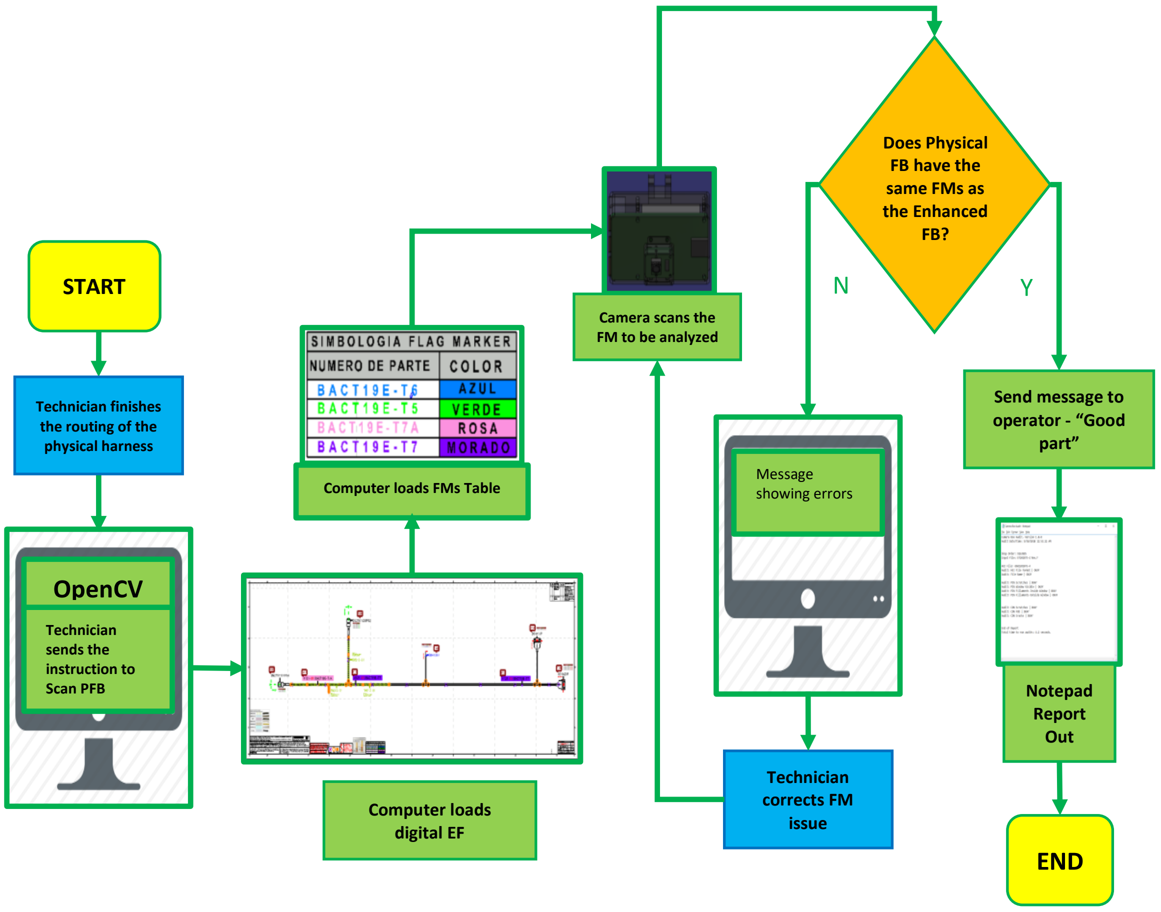


Figure 3.2: Proof of concept of how inspection system would function in a real manufacturing setting. The system would inspect sections of the wire harness that have already been assembled by the worker.

With an understanding of some of the challenges, the problem was focused into a series of requirements that the system had to be able to execute. With limited experience in automated visual inspection, these initial guidelines were used a base line for the case study. We will see how these initial requirements helped to both better understand the problem and develop new and more essential requirements. The requirements are as follows:

1. The system needed to differentiate between humans and key features of interest in the image in order to ensure that the technician was finished working or not
2. The system had to be able to accurately detect features from a maximum distance of 4 ft. This distance was based on the layout of the facility.
3. The system had to position itself automatically without the need for interaction of another worker. Ideally, once the information for the part was loaded, a single button push would automatically start the inspection process once the technician reached a certain point in assembly
4. System needed to be relatively cheap and easily implemented into the current manufacturing environment. There would potentially need to be hundreds of these cameras and while there were off the shelf camera systems, most were expensive and not easily implemented.

With these requirements in mind, efforts were made to investigate the feasibility of implementing a system that could meet them. A flow chart was developed in Figure 3.3 that shows the logical progression of the process. Once the technician finishes assembling the wire harness, a computer would load the requirements and specifications for the specific FB needed for the harness. Afterwards, the harness would be scanned for correctness and a report would be distributed indicating success or failure of assembly.



START
Technician finishes the routing of the physical harness
OpenCV
Technician sends the instruction to Scan PFB
Computer loads digital EF
Computer loads FMs Table
SIMBOLOGIA FLAG MARKER
NUMERO DE PARTE
COLOR
BACT19E-T6
AZUL
BACT19E-T5
VERDE
BACT19E-T7A
ROSA
BACT19E-T7
MORADO
Camera scans the FM to be analyzed
Does Physical FB have the same FMs as the Enhanced FB?
N
Y
Message showing errors
Technician corrects FM issue
Send message to operator - "Good part"
Notepad Report Out
END

Figure 3.3: Wire harness inspection flow chart. FM means flag marker, a feature of interest in the detection, FB is the form board, and EF is electrical figure for the drawing. Flag markers will be discussed in the upcoming paragraphs.

This interdisciplinary project required investigation into computer vision, image processing, mechanical design, and later, control and image-based control. In the next section, the efforts made into each of these areas will be explored as well as how they helped in tuning our requirements to better solve the problem. It is important to note that each of the steps taken led to the main research goal of this thesis: to develop an optimal and robust vision control techniques to be used for precision visual inspection.

3.3. Inspection system initial prototype

The computer vision and image processing aspect of this project will be discussed first. The first step was deciding on a camera. Table 3.1 below shows all the cameras that were considered below along with the pros and cons.

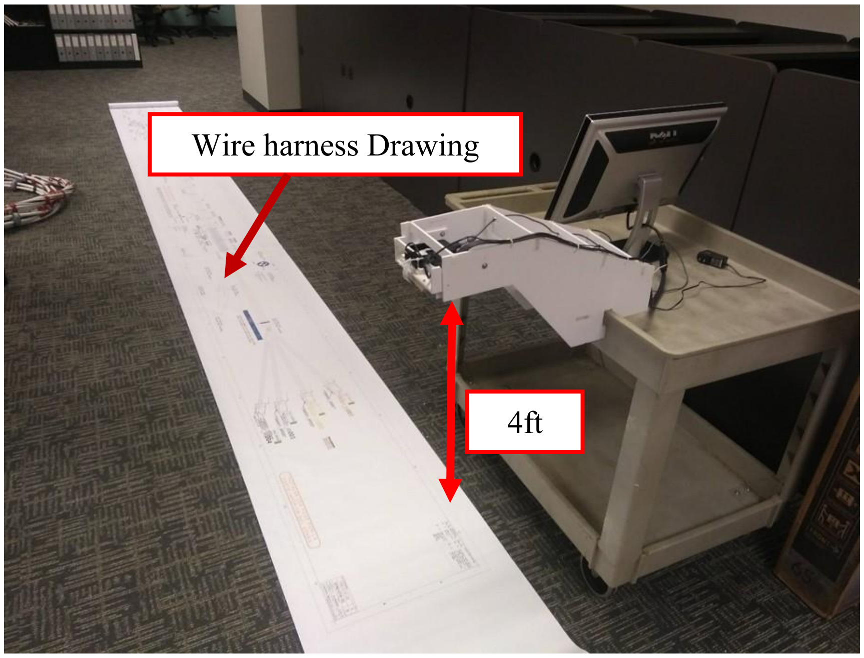
Table 3.1: List of potential cameras that were available to use for image capturing and processing.

Camera Options for Inspection System		
Camera Type:	**Pros:**	**Cons:**
Raspberry Pi Camera Module V2	• Software supported with opensource software platforms • Good with low light photography • Option to make a graphical user interface (GUI) with software available to raspberry pi micro controller • Lab has worked with this camera for other projects	• Cannot reach as high of camera quality as other cameras (max resolution is 1080p) • Limited capability for video
Sony IMX219 for Raspberry Pi	• Capable of high-resolution video as well as high quality image capturing • Usable with raspberry pi microcontroller	• Would have to develop a case for the system as it is a new camera • More expensive for features that may not be necessary to solve problem
Pixy (CMUcam5)	• Open source however interfaced easier with Arduino controllers • Built in object recognition for a variety of colors	• Built in recognition software is limited • Would have to develop a case

Picking a camera was critical, as the camera would have to be cheap, able to take pictures at a high enough resolution to process the image and be able to relay the image data for post processing of the image on a microcontroller of some sort. The camera that was decided on was a raspberry pi camera that raspberry pi offers as an add on attachment to their microcontrollers. The main reasoning behind this decision was how much the lab had used this camera in the past. There were a lot of resources to develop computer vision algorithms using this camera as well as a variety of cases and setups that could be used for building a physical system.

Once the camera was decided upon, a few prototype systems were developed to hold the camera. These systems were mainly used to run initial tests of the computer vision code used for analysis by the raspberry pi. The raspberry pi microcontroller offers a wide variety of onboard processing and has the capability to programmed in Python. OpenCV, an open source computer vision software, has Python bindings that allow it to easily be used for various computer vision applications where the controller can be programmed in Python. This open source capability also allows us to utilize it on another system used to evaluate the performance of IBVS and DP and will be discussed later in this thesis.

The first prototype of the system used to hold to camera can be seen in Figure 3.4. Utilizing this prototype, there were a lot of critical elements learned about the image capturing and image processing that aided in the design of the final system used; specifically, addressing point 2 mentioned in the previous section of capturing images at 4 ft.



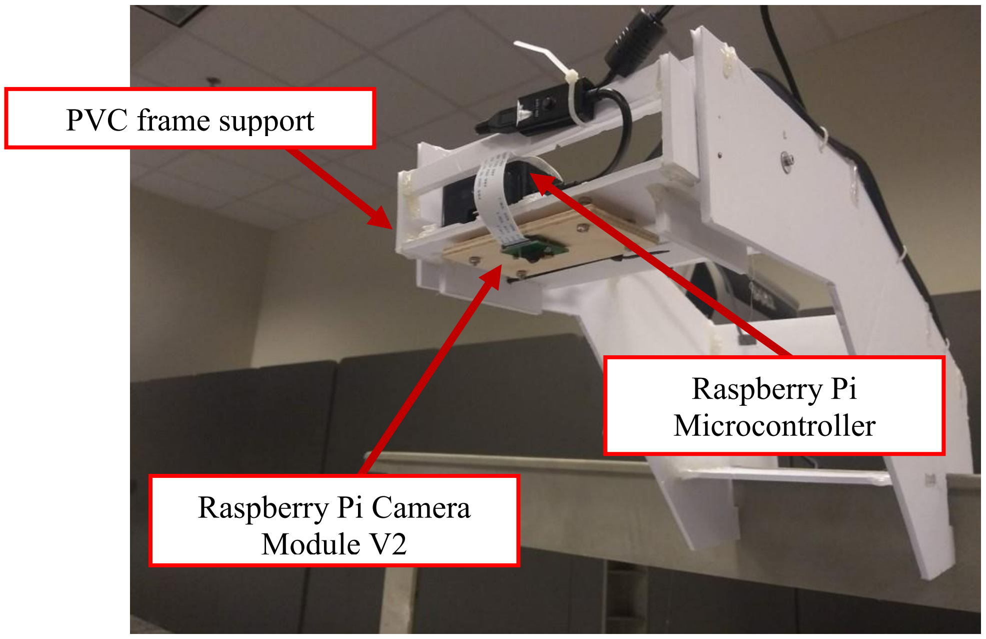


Figure 3.4: Prototype system developed for initial testing of code. The camera was placed 4 ft from the wire harness drawing as that was the estimated distance available for image capturing

The structure was made of PVC as its only function was to hold the camera and create a constant distance of 4 ft between the drawing and the camera. A raspberry pi micro controller with the chosen camera module was connected to a monitor for programming using OpenCV. The details of the lessons as learned and how the address point 1 in the previous section will be discussed in the next section.

3.4. Image Processing and OpenCV

As stated in point 1, it was determined that the system would need to accurately be able to distinguish between humans and the desired features of the wire harness. This was to ensure that the inspection system was inspecting the correct place of the wire harness. If there was a human inside the frame, this would mean that the human was still assembling the harness. Keep in mind that the objective of this system would be to inspect section that have already been assembled by the technician. Using the raspberry pi camera module and some basic human detection algorithms in OpenCV, we explored the idea of human detection to see how bust we can make the algorithm. Figure 3.5 shows the results of the human detection and the confidence rating the algorithm calculated to show how sure it was that it was a human in the picture. The images shown in the figure are snapshots of live video feed. This mean the detection algorithm was detecting humans in real time and not post processing the images. It was determined that if possible, a live video feed would work best for the inspection system.

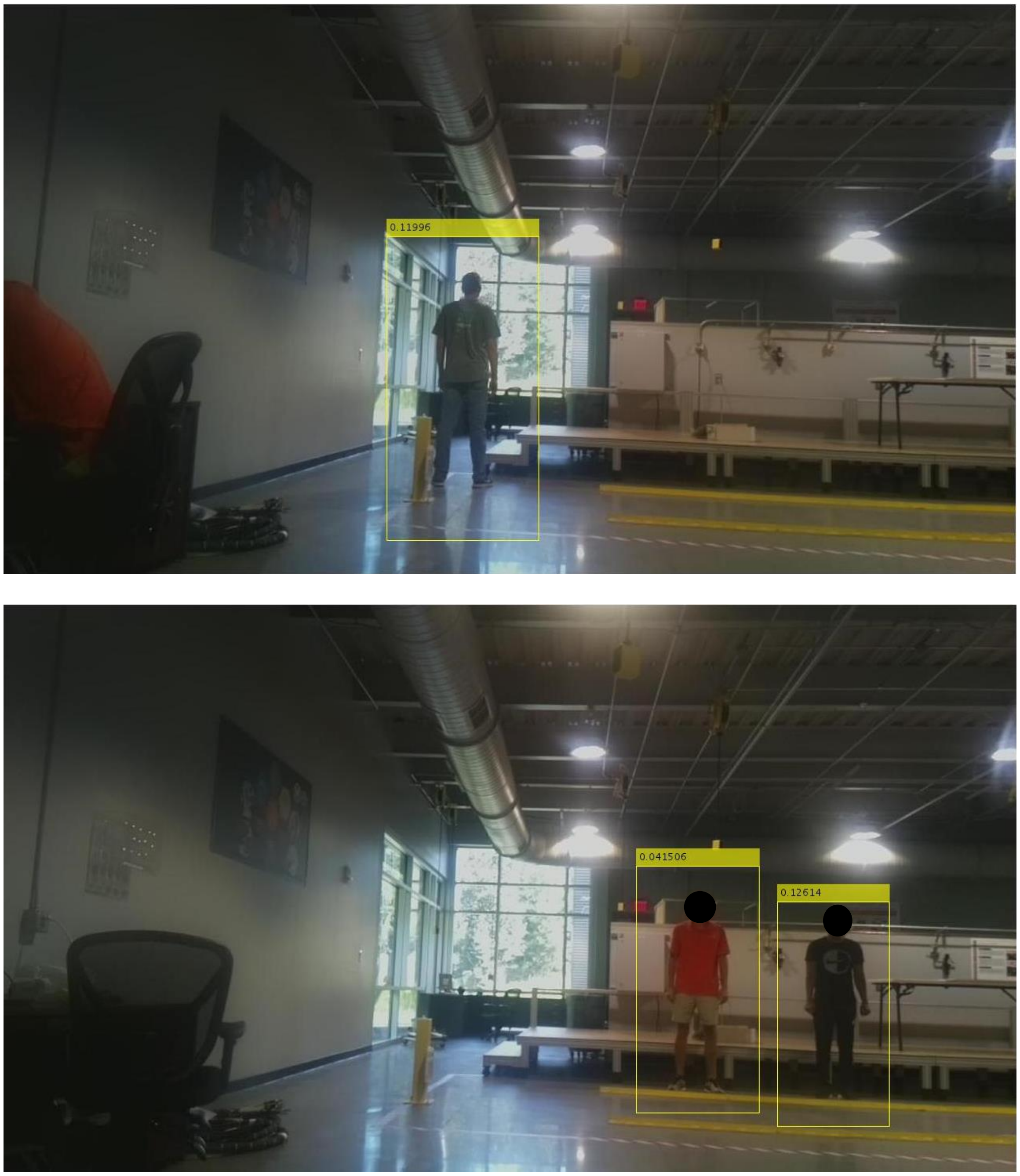


Figure 3.5: Human detection with confidence intervals. The algorithm was able to detect more than one human in a frame; however, confidence went down as more humans entered the frame.

The algorithm worked well at determining if there were humans in the frame, however there were a few things discovered during this test. Recording video required a lot of processing power for our microcontroller to record at a high enough resolution to detect humans. The image above is a snapshot of a video feed taken at 1080p30, which is the highest resolution possible by the camera. Even at this resolution however, we can see that the confidence was relatively low. In the first image, the algorithm is only 11% sure that the person being shown is a human, and in the second image, it is 4% and 12% sure respectively. This cannot be seen in the image, but during the test, the yellow box can be seen going in and out, further indicating the uncertainty of the algorithm that the image in the frame was a human.

Understanding that the algorithm must be able to confidently detect humans in the camera frame, attempts were made to increase confidence, however none of the methods worked well on post processing of an image as this would need to be done in order to reduce the processing power required for a video feed. In order to reduce the processing power, the algorithms would need to capture and store images every couple of seconds to check for humans in the camera frame which would rapidly deplete storage. The storage would need to be used for storing images during the actual inspection process of the system. The last attempt made to make the human detection better was to delete the photos as they were deemed to not have a human in the image. This solution also did not show consistency in confidence as there were still cases where a human was present, and no detection would show up. Despite the human detection algorithms not being robust, fortunately the detection of flag markers, a key feature of interest in inspection, had simple geometry as well as bright, easy to capture colors that allowed the algorithms for feature detection to work with consistency.

Flag markers are used as an indicator for SAFRAN to identify key sections on a wire harness. Below in Figure 3.6 shows a picture of test wire harness that was used for developing the system.

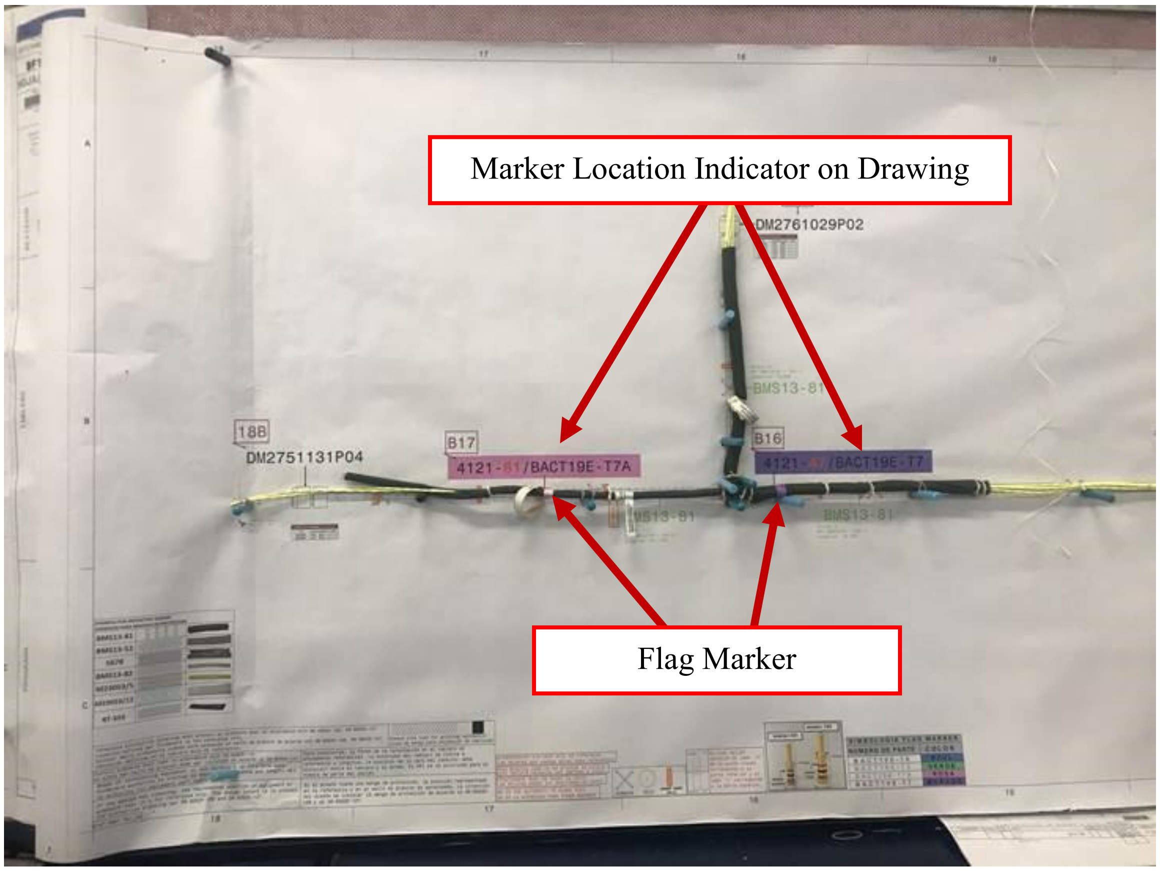


Figure 3.6: Flag markers on wire harness used for testing. Not a real part used in production but has the shape, drawing, and orientation of a real wire harness.

This wire harness was donated for testing purposes; however, it closely resembles the wire harnesses used in production in the plant. In the figure there are two key components: flag marker indicator on the drawing which sows there the marker should be placed on the harness, and the physical marker that is placed on the wire harness. It was decided that the flag marker would be the first feature of interest to attempt to detect due to how simple yet common the indicator appears on a wire harness. Some wire harnesses can have 20-30 flag markers over the course of a 10-meter span. The technique used for the detection of the flag marker was an object recognition computer vision algorithm that searched for both shape and color. Fortunately, the shape of the flag marker on both the drawing on the harness are simple squares. In Figure 3.7 we can see the detection system tested on three different flag markers. Notice how cluttered the image is with objects of different shapes and colors in the image frame. Despite this, the algorithm still can accurately determine that a flag marker was present in the image frame. In testing we noticed that even with a human in front of the image plane, the flag marker was still able to be detected thus eliminating the need for a human detection, which was not consistent in human detection anyway. This helped fine tune the requirement listed in point 1 to not include the need to detect humans in the image frame. While the algorithm consistently and accurately detected the features, there will a few issues with color differentiation between colors that were close in shade; specifically, the colors blue and purple were difficult to differentiate between. This could have been improved however we focused on colors that were further in shade.

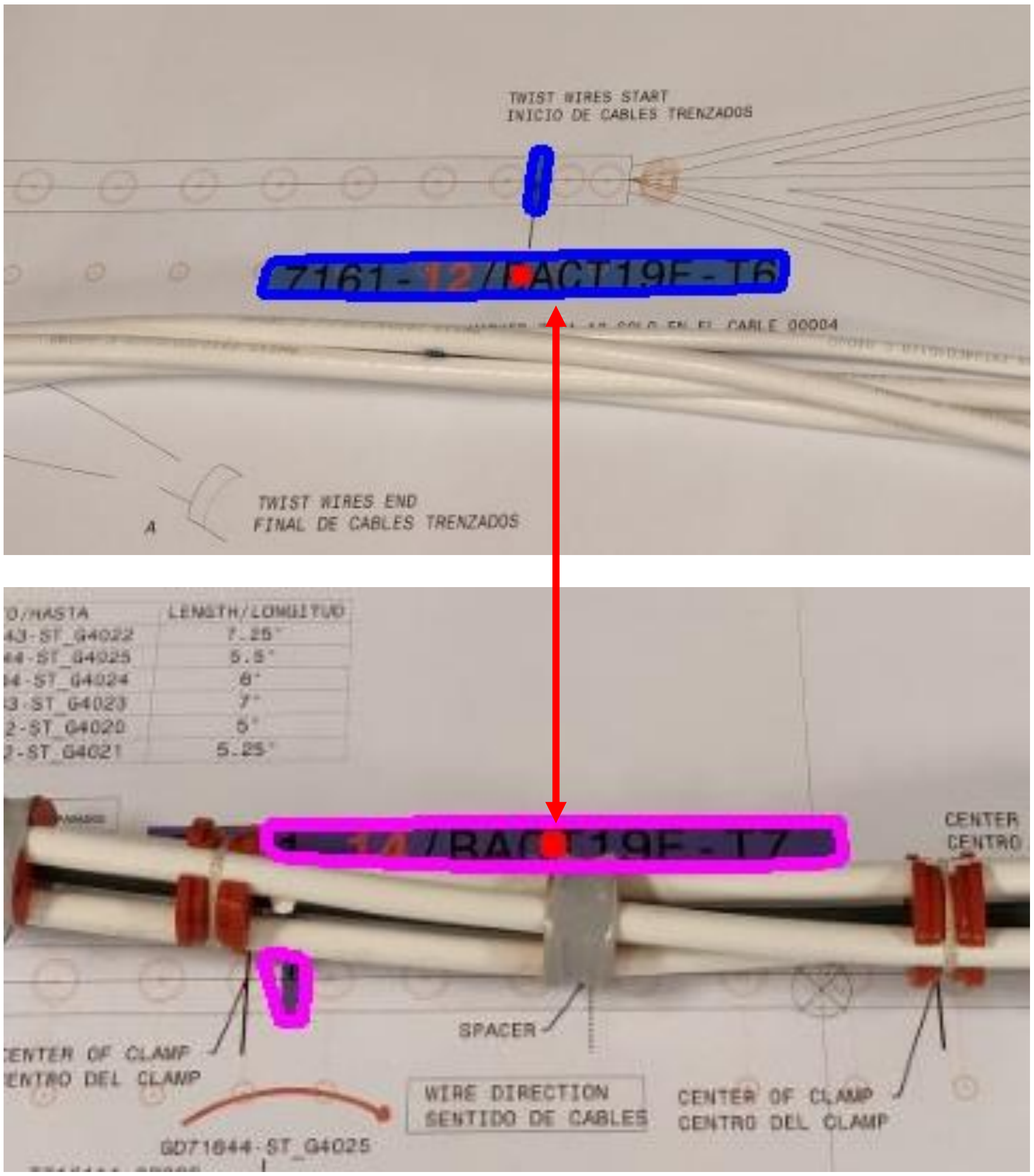
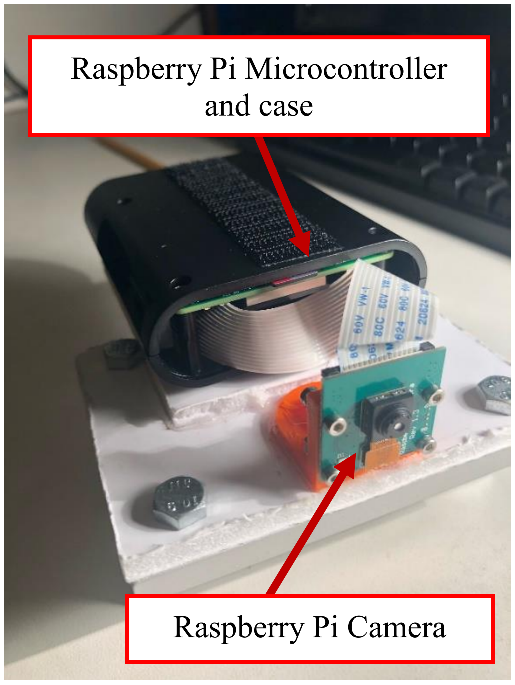


Figure 3.7: Flag marker detection algorithm detecting blue and purple flag markers. In this specific image, the flag marker that is physically placed on the harness is not shown, however we will see in another image that the marker can also detect the physical flag marker on the harness.

From the image detection software, we were using, we were able to successfully detect a feature on wire harness known as the flag marker. We were able to successfully address points 1 and 2, modifying point 1 due to the accuracy if the feature detection algorithm used to get detect the flag markers. The next aspect of the issue was positioning and controlling the system to accurately and scan and inspect the system for part correctness. The current prototype was useful in testing feature detection algorithms, however we needed to move to a system that could be placed onto a track above where the workers assembled the harness. Investigation began on a designing and building a track type system that could hold the raspberry pi camera and micro controller. Figure 3.8 shows the camera case used to mount the raspberry pi on a track system. The mount was 3D printed and bolted onto a platform designed to mount to a railing track fixture. The design of this portion of the system was meant to be simple and effective. There were several ways the camera could have been set up for a track system. There were considerations of a variety of linear actuators, guide rails and track systems, and rotary shaft systems, however we were not interested in the design aspect of solving the problem, but rather developing robust algorithms for motion control and image detection. This simple design was effective and allowed us to attach the camera mount to a track with a timing belt and stepper motor and servo along a track with ease. The case used to house the microcontroller was a standard case that came stock. It was modified with 2 holes in order to mount correctly to the guiding track.



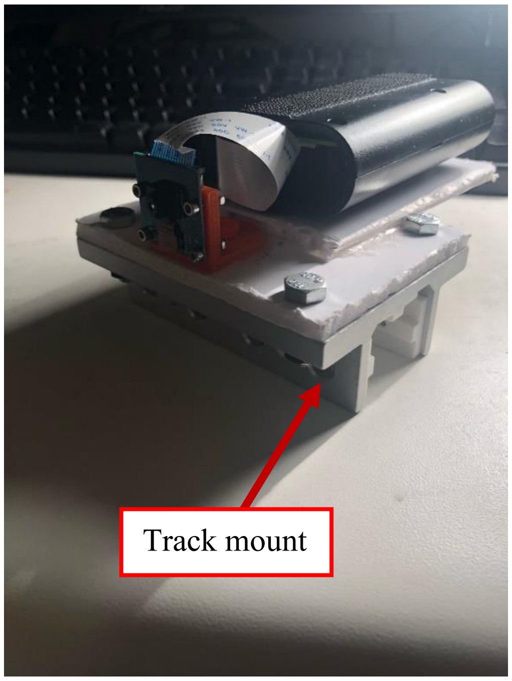


Figure 3.8: Raspberry pi camera attached to raspberry pi microcontroller bolted onto a track mount to be attached to a guiding track system

A guiding track frame was developed in order to synchronize the image capturing process with motion control. Given the lengths of some of the form boards, we needed a test system that was big enough to scan multiple features. The system that was built had a 4ft by 8ft section where the wire harness drawing, and physical wire harness would be mounted. The track was placed around 4ft away from the board to simulate the real working environment in the plant. As stated before in the design of the camera mount, the designing process of this structure was not the focus. A simple design was used to isolate both the form board and the track. A CAD model of the structure can be seen in Figure 3.9.

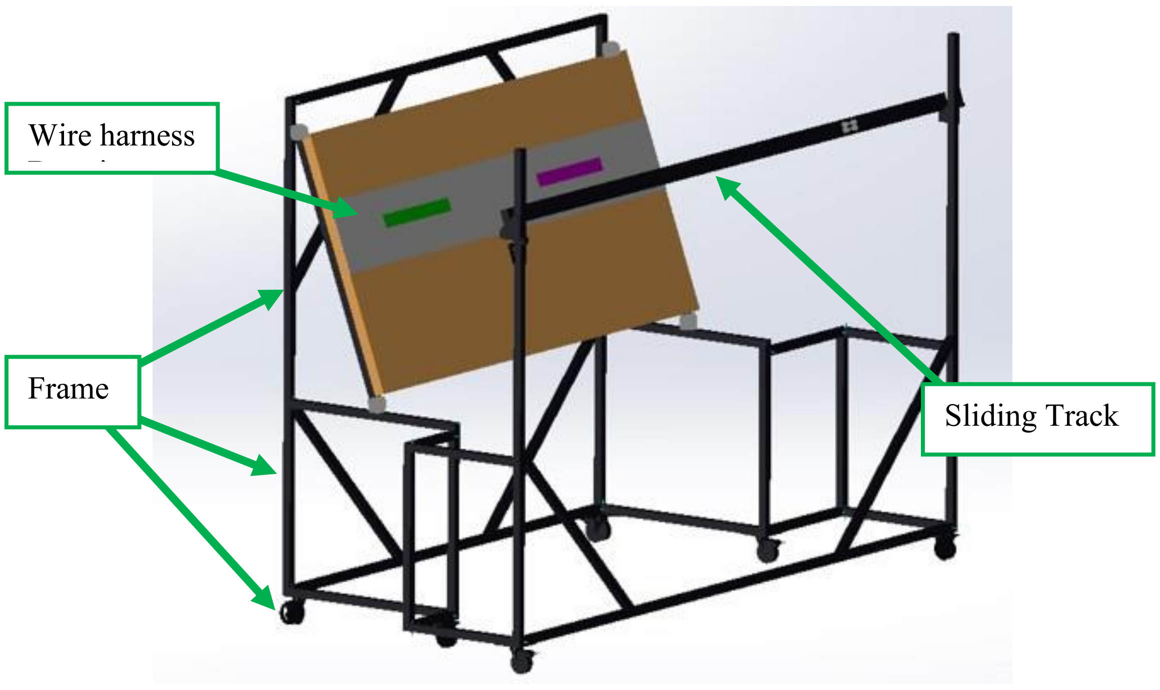


Figure 3.9: Solidworks mode of set up for evaluations of airplane wire harnesses. Model is not a full-scale wire harness. A portion of a full-scale harness was used for testing inspection of specific features.

The was frame built from 80/20 extruded aluminum. The raspberry pi subsystem is attached to the track and can slide along the length of the wire harness. The camera is actuated along the track using a bipolar stepper motor programmed by an Arduino microcontroller. At the time of fabrication, several difficulties were unknown and limited the robustness of both servoing the camera and performing the image processing. This led to the investigation of image driven servoing. In the next section, examples of the image capturing process with the full system along with collected data of the accuracy of some of the motion control will be shown. This data is what motivated to use for exploring image driven motion control strategies.

3.5. Motion Synchronization

As mentioned in point 3, the system would have to automatically perform the operation with limited intervention from the technician. From the flowchart in Figure 3.3, a single button push should initiate the sequence of actions needed to complete the inspection. Figure 3.10 below shows the full, fabricated system in action inspecting a green flag marker on the wire harness.

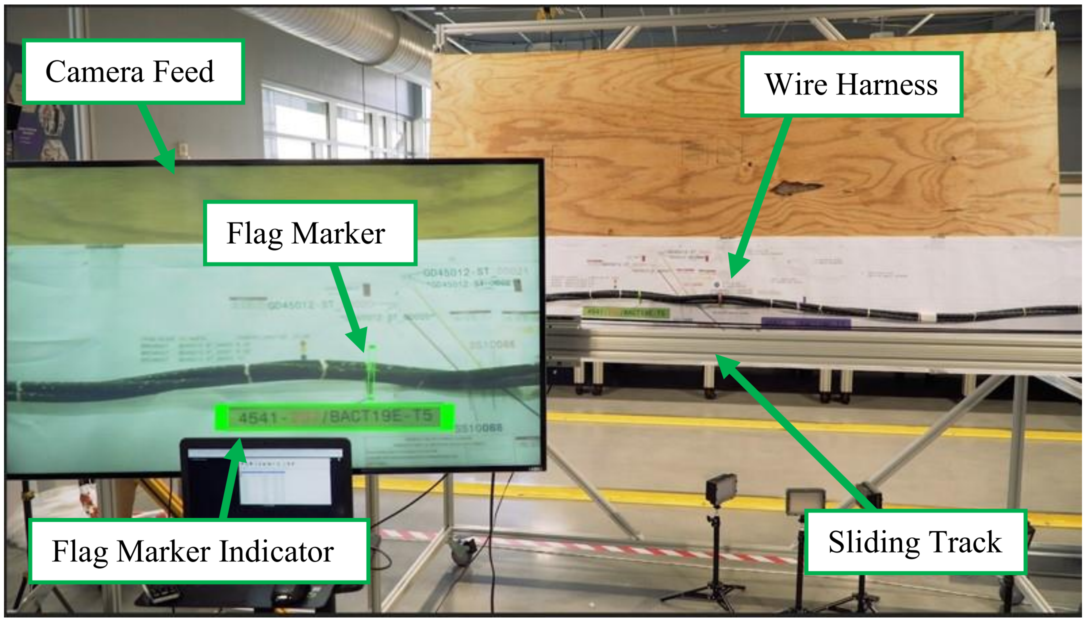


Figure 3.10: Physical set up of inspection systems with visual representation of object recognition. While a live camera feed could be seen during operation, it was not recording and only storing images once the algorithm was initiated. This was to save storage on the raspberry pi.

The motor was set to move the camera at a rate of 3 in. per second based on the 4 ft. distance between the camera and form board. This allowed for a seamless series of photos to be captured along the wire harness. The robustness of the feature detection allowed the camera to identify features with the motor moving at this rate. Initially, it was thought that the feature detection and the motion of the track could function independent of each other if they were initiated at the same time. The experiments that were conducted however, show that despite knowing where the object is, often the feature detection will not detect the image until after the system has already passed the feature. To measure the performance, a graph was developed to track when the snapshot of the feature was taken. This can be seen in Figure 3.11.

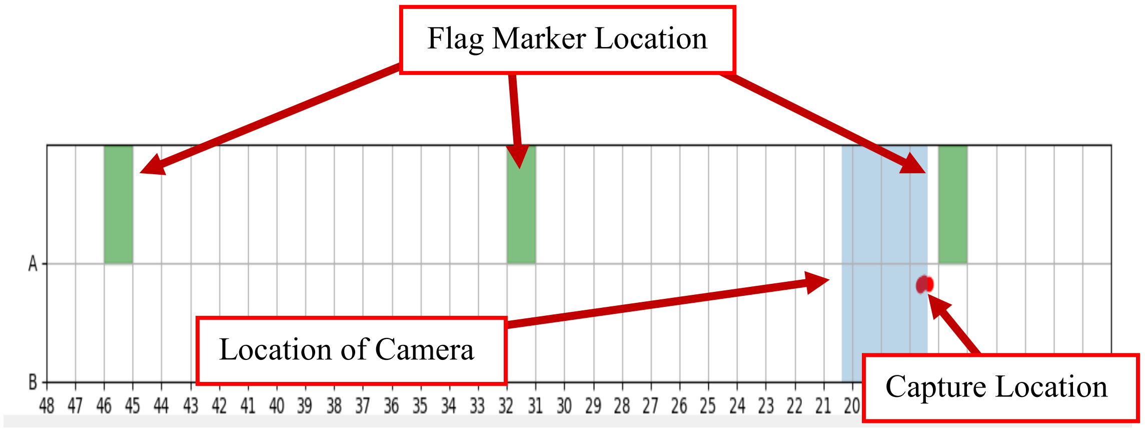


Figure 3.11: Graph of feature detection accuracy. This setup combined the feature detection algorithms used prior with basic motor control of a stepper motor.

On this graph, each section represented roughly a 2-inch window. It can be seen from Figure 3.10 how small the physical flag markers are on the system. The algorithm first detected the indication of the flag marker on the drawing and then, using the at information and location, tracked the location of the physical flag markers. The green rectangle represents the location of the flag marker. This was calibrated to be the location where the flag marker would be in the center of the screen. The red dots indicated the position where the camera detected the feature. There are multiple dots clumped together due to the feature detection activating and deactivating. Once the algorithm detected the feature three times, it was stop looking for it. Note that the feature was still detected, however it was not at the location that it was expected to. There are a few potential reasons for why the feature detection location was off. One reason was the velocity that the camera was translating. While most of the images to the human eye were clear enough to identify the flag marker, we noticed that images that were taken completely still were slightly clearer than those taken while in motion despite the velocity being relatively slow. Another issue was the distance that the camera was taking the images. In Figure 3.10 you can see how the flag marker indicator takes a large portion of the image. Being that the marker must be centered for the capture to be considered aligned, if the feature was not entirely in the camera frame, it would not detect the feature accurately. There were attempts address some these issues by using the prototype system in Figure 3.4. The results of those can be seen in the graph below in Figure 3.12. Note that due to the translation being manual, there is a slight variation in the depth of the image capturing. We did not track depth on the graph, so the numerical variance is unknown, however it is slight and did not impact the feature detection.

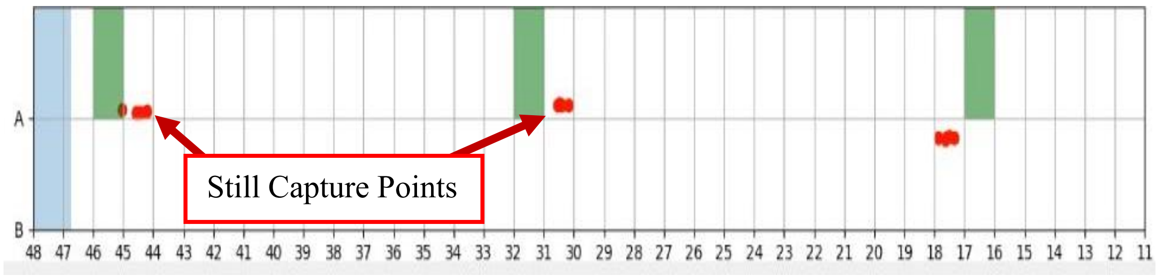


Figure 3.12: Detection algorithm results of prototype system. Used to determine if the capturing could be improved with manual variation of the translation speed.

The first point was captured at a speed that approximated to be around the 3 ips velocity of the automated prototype system. The specific value of the velocity was not important; however, it was necessary to have a velocity that was relatively close to the velocity of the automated system. From the first point on the right, there is still the issue of the feature not accurately being detected in the correct space, which was expected. The two points on the left are cases where the camera was slowed down to almost a complete stop while translating to capture image. The feature detection began to correctly identify the features in both cases before the camera was in the specific location of the feature. Some trials were done to test the repeatability of what was happening when scanning the flag markers in this way. Table 3.2 below shows the results from those trials as well as the percentage of offset the captured image was from the actual feature. The flag markers are ordered from right to left (i.e. flag marker 1 represents the right most green marker on the graph). As mentioned before, each of the columns on the graphs represent a 2-inch section.

Table 3.2: Data taken of flag marker location compared to flag marker detection. 3 trials were run for three different flag markers along the formboard.

	Trial 1	Trial 2	Trial 3	Avg.
Flag Marker 1				
Location [in]:	32	32	32	32
Capture Location [in]:	34.8	35	34.9	34.9
Percent Difference:	-8.05%	-8.57%	-8.31%	-8.31%
Flag Marker 2				
Location [in]:	62	62	62	62
Capture Location [in]:	58.8	59.5	60.1	59.4667
Percent Difference:	5.44%	4.20%	3.16%	4.27%
Flag Marker 3				
Location [in]:	90	90	90	90
Capture Location [in]:	88.1	88.4	88.4	88.3
Percent Difference:	2.16%	1.81%	1.81%	1.93%

The negative percentage offset represents the flag marker being detected before it was supposed to while the positive percent offset represents the flag marker being detected afterwards. For the first flag marker, it was expected that the feature would be detected before the marker was detected given the results from the flag marker detection on the automated prototype system. Quantifying it shows an offset of about 8.31%, which is relatively high for the aerospace industry. Even in the cases where we slowed the translation of the camera down dramatically, the percent offset of detection is still considered high for the aerospace industry. SAFRAN typically has tolerances that are less than 1% when manufacturing wire harnesses. For the flag marker specifically, an offset a few inches could mean catastrophic failure of the plane.

A few things were learned in this section about the motion synchronization. The first is that having a separate system for the motion of the motor and the image processing works well in applications where slight offsets do not affect the interpretation of the data. In the case of detecting flag markers, these offsets are too large given the strict requirements of the aerospace industry. The second lesson was that image quality is affected when the image is being captured in motion even at slower speeds. The reasoning behind using a velocity of 3 inches per second was that it was assumed it was slow enough to capture the images without blurring the image enough for the feature detection algorithm to still work. The natural line of thought after these results would be to have the motor stop to capture the flag marker. Being that the location of the flag marker is known in most cases, this seems like it would work well. From Table 3.2 we can see this is not the case. While this test was manual done, flag marker 1 and 2 were detected earlier than the predicted location and still had percent offsets that are considered high in the aerospace industry. It was critical that the flag marker is detected in the exact spot that it is supposed to be. An idea that was never tested due to limitations was having the camera stop at specific points while having a high translational velocity while translating between points. The use of higher speeds would prevent the algorithm from activating due to the camera not being able to recognize the flag marker at a high speed. It important to remember that initially it was thought the motor and the camera would operate separately. This was due to the large distances the camera would have to travel in a real manufacturing environment. In this scenario, the slower velocity would allow a scanning type of procedure to assess correctness of the formboard. While a high and precise velocity experiment was not conducted due to time, this type of control would be impractical in a real manufacturing setting despite being useful in the controlled environment.

There are several ways to perform feature detection and it is known that further investigation into other image capturing and processing methods could yield better results. The more time on this project as well as a deeper background in computer vision would aid in understanding ways to improve image capturability and processing. Despite these setbacks, there are still valuable lessons and gaps in technology that were discovered through this case study if automation is to effectively enter the aerospace industry. The main takeaways are:

- Precise robotic systems are necessary if automated assembly is going to enter the aerospace industry. Systems that cannot prove consistency, high precision, and are easily to implemented will never be considered.
- Having separate controllers for image capturing/processing and motor controllers requires more advanced computer vision algorithms for high precision applications, however the computer vision algorithms used in this work perform well for most applications in commercial use as well as industries with more lenient tolerance requirements.
- Techniques using motor control based off image data would work best in applications where precision is important. Being able to ensure the camera is in the

right position based on how it appears in the camera frame ensures that the feature detection is confirming the location in the right place.

The results from this work motivated the investigation of image-based control techniques, specifically IBVS. These techniques could guarantee consistency in the detection as the feature detection could operate separate of the control. Chapter 4 will go over the simulations that were conducted with IBVS as well as the combined IBVS and dynamic programming (DP) control scheme.

3.6. Summary

In this chapter, a case study was conducted to illustrate where the motivation to investigate image-based control techniques comes from. The aerospace industry is looking to follow the trend of automation and is starting by investigating the area of quality inspection. This sector of manufacturing in aerospace is critical as parts need to be assembled to a tight tolerance to avoid catastrophic failure. SAFRAN, a leader in this industry, was interested in automating the inspection process of airplane wire harnesses. Given the size and complexity of the harness, a mistake in the assembly process could take a long time to both find and fix. The first feature of interest was detecting the flag marker, and indication measure used to indicate where on the airplane a specific section of this harness needed to be located. Fortunately, this feature specifically had a simple shape and is bright in color, opening the door for a wide variety of feature detection algorithms to be used. OpenCV was the program of choice used to implement as it could programmed on a raspberry pi microprocessor using python. Being able to use the microprocessor was critical as it is cheap and compact. Once the feature detection is consistent enough in the image detection, the next step was determining the best way to actuate the camera along the form board to detect flag markers. What was initially thought to be the easier aspect of the project turned out to be the area where further investigation needed to be done. The original approach was to have the image capturing and processing operate independently of the motor control (i.e. having the motor move a set velocity and having the camera detect features as it passed them). While the features were detected, it was difficult to detect them precisely and consistently while the camera was in motion. Precision is critical given the tolerances that are typically allowed in the aerospace manufacturing field. Some reasons for this lack of robustness were the computer vision algorithms, the speed of the camera, and accuracy of the motion control.

The topic of computer vision is a broad one. While there may be several more advanced computer vision techniques to explore, this was not the aim of the work. It was deemed more beneficial to explore image-based control methods that would ensure precision motor control of the camera. Using these methods would allow us to keep the feature detection algorithms which already work well and focus on the motor control. As mentioned in Chapter 2, there are several methods of image-based control methods to explore, however it deemed that IBVS would be the best for this application. It is relatively simple to implement in cases where high precision is not required and is used in a wide variety of manufacturing environments already. There are still setbacks that will be explored in the next chapter. These setbacks limit the precision required for implementation in an aerospace manufacturing environment, however these limitations will be addressed using DP. The algorithms for human detection as well as basic edge detection algorithms used for flag marker detection can be found in Appendix B.

CHAPTER FOUR

Simulation of IBVS and combined IBVS and DP control scheme

This chapter will go over the details of simulating IBVS and the shortcomings. As we saw in the last chapter, image-based control techniques could solve the problem of imprecise feature detection and control. This chapter is outlined as follows: detailed explanation of initial simulations of combined DP and IBVS, robust IBVS simulation with shortcomings, DP and IBVS combined control scheme addressing shortcomings, and lastly, simulating the combined DP and IBVS control scheme on a for a scenario likely to occur in the real world.

4.1. Initial attempts at combined control scheme

The first simulation was attempting to program the control structure in MATLAB based off the equations listed in Chapter 2, specifically Equation (24). In the simulations employed to demonstrate this combined approach, DP is applied in a recursive manor using Equation (24). This equation is used to minimize the time required for the camera to reach the desired pose, which in turn minimizes the distance traveled. At a high level however, the process works using the principle of optimality outlined above except for instead of 1 point, there are multiple. In our simulation, it was decided to track what was assumed to be the minimum amount of points necessary to simulate our equation. This was 3 points as it was the minimum amount of points needed to form a shape (a triangle in this case). This because the interaction matrix must be at minimum a square matrix in order to be properly calculated. Due to the image coordinates inputted into the interaction matrix for one point having dimensions of 2x6, in order to run dynamic programming, a minimum of 3 points would be required to result in an interaction matrix that is a square 6x6 matrix. We later use IBVS with more than three points, however due to the need to calculate the control input for dynamic programming in this initial attempt, the interaction matrix L_s ended up being a square matrix. As outlined in Chapter 2, when using dynamic programming, all the possible paths must be considered in order to determine the minimum cost. When there is only one point, this iteration process is relatively straight forward, however with three points one must consider the optimal paths to go to each state for each point. This means that the minimum cost for one point may imply a direction that is not the minimum path for another. Once an optimal path is determined, the optimal path for the other two points must be determined based on that point. If the optimal path for the next point cannot be determined, the calculation must start over and go to the next minimum path. The number of iterations can go up exponentially and require a lot of computations for large translations and for image spaces with many potential states. This bring up the importance of correctly choosing the state variables. We can get around this by tracking the rotation of the system instead of the coordinates for each individual point. We will see this in a later example.

In the next section, a simulation is shown as well as an example of the limitations of IBVS. Some modifications to the code were necessary to extract the data during the simulation.

4.2. IBVS example

An IBVS example was developed to understand how the combined IBVS and DP control scheme could implement into it. One difference in this algorithm was that it required 4 points as opposed to 3. This required modifications of our combined scheme by adding another 2x6 vector for the fourth point, making the interaction matrix an 8x6 matrix. The scale of pixel coordinates was also increased. This did not change anything related to our algorithm, however the scale of the cost values calculated in the later sections was subsequentially also increased. As mentioned in Chapter 2, rotation about the z axis is typically what causes issues with IBVS. We first look at an example where the algorithm successfully performs a rotation. We start with a square 400x400 pixel square where the desired camera pose is directly centered on the screen. The image space is partitioned into a 1000x1000 pixel space. Being that there are four points and the desired pose is directly centered onto the screen, coordinates of point 1, 2, 3, and 4 are $(u_1, v_1) = (300, 300)$, $(u_2, v_2) = (700, 300)$, $(u_3, v_3) = (700, 700)$, $(u_4, v_4) = (300, 300)$, respectively. For the rest of these simulation, the desired camera pose will be based off these image coordinates. It also assumed the focal length λ is equal to 1. The initial camera pose is simply rotated about the *z* axis by 1 radian, or 57°. The trivialness of this case is intentional as the simulation should converge onto the desired camera pose without any problems. While we will see this, we will also notice how even in this somewhat trivial case, there are still issues with the camera taking an unusual path in the *z* direction before converging. While this is a slight detour, it is still concerning in the implementation on a real system given the level of precision required. Figure 4.1 below shows the simulation of this translation. The square with the stars represents the desired image orientation and the square with the circles represents image orientation. The blue line connecting them represents the path taken to reach the desired camera orientation.

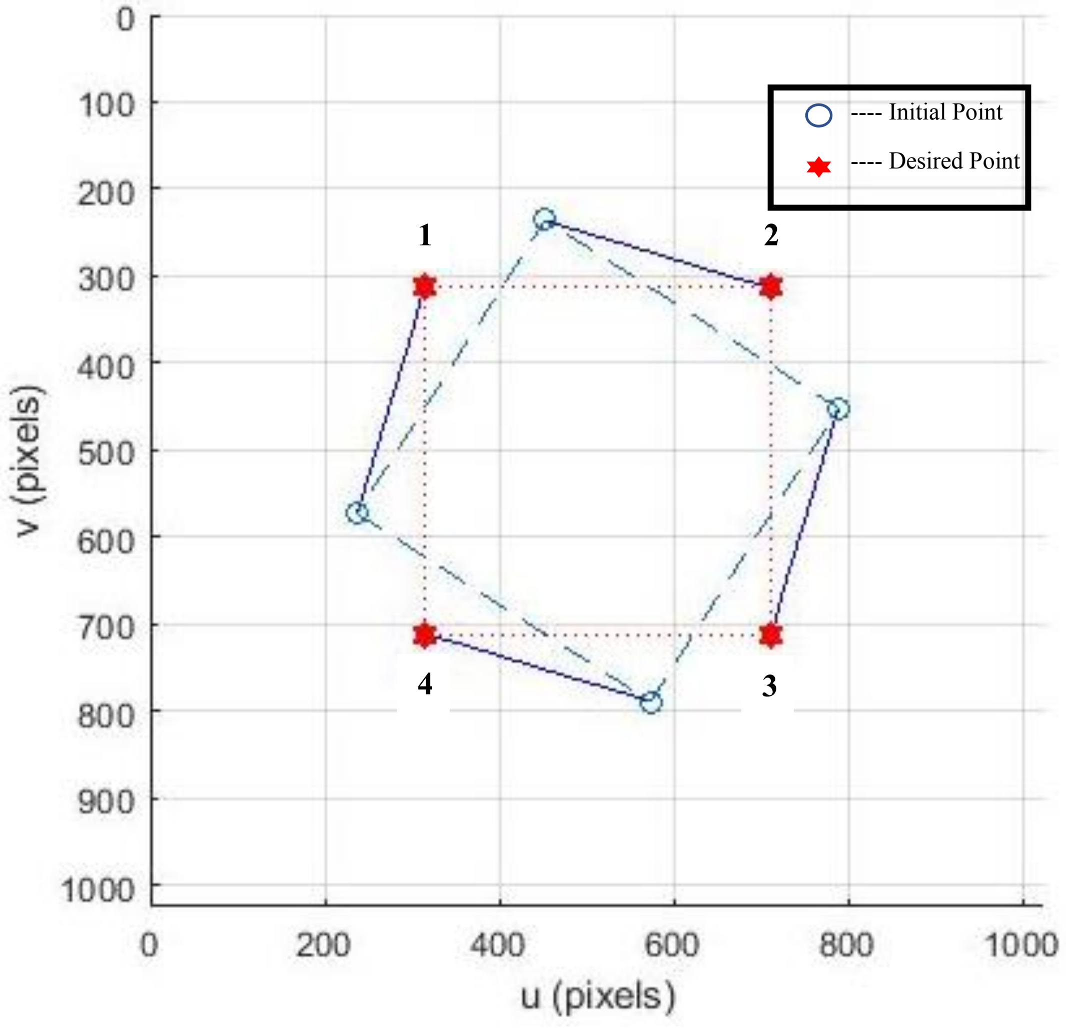


Figure 4.1: Case of IBVS performing rotation about *z* axis and successfully converging following the most optimal path to reach the desired trajectory.

Running this simulation was important to understanding how IBVS could be run. One thing this simulation result doesn't show however, is the motion in the *z* direction. A graph was generated to track both the translational and rotational positions of the system. The graph of both the translational camera position and the rotational camera position can be seen below in Figure 4.2.

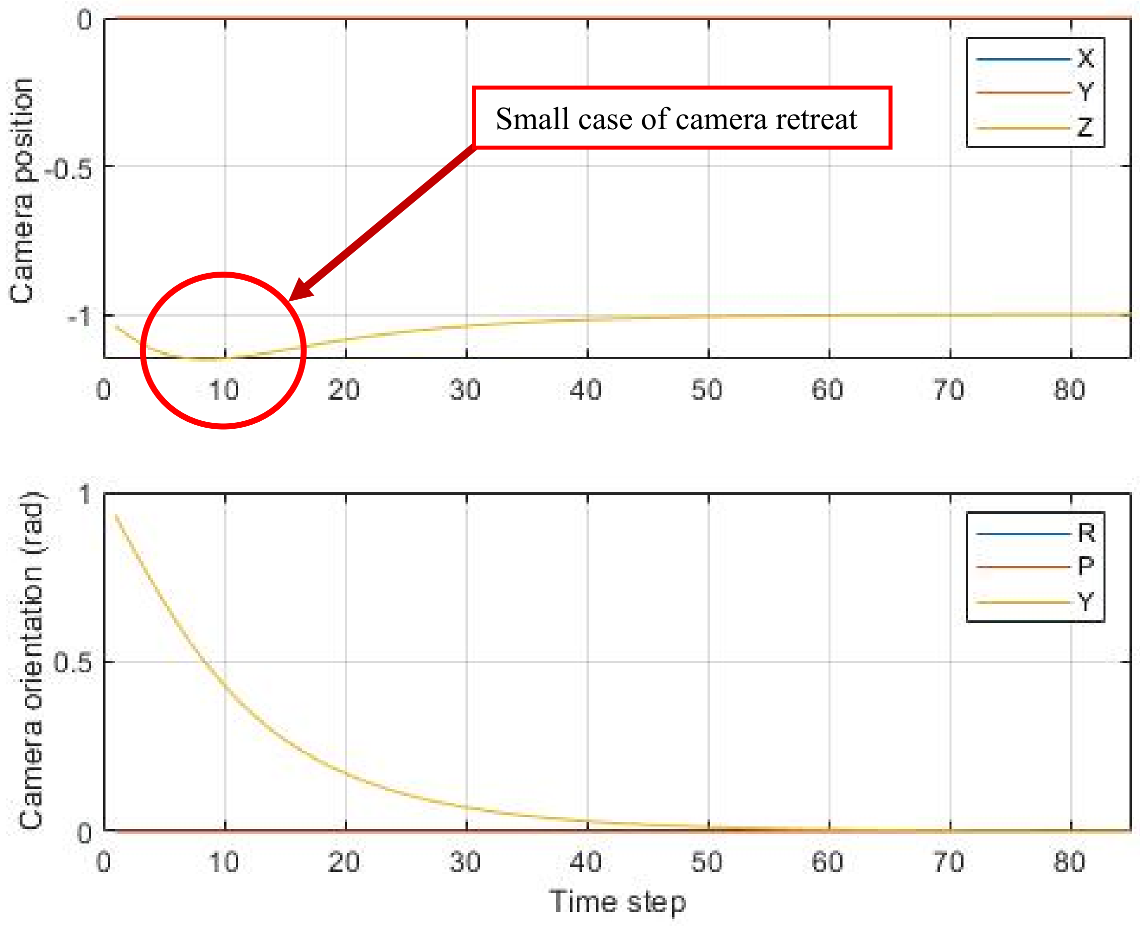


Figure 4.2: Camera translational and rotational orientation. The *X, Y,* and *Z* represent the camera position. The depth of the simulation was set to 1 and since the motion was purely rotational, *X* and *Y* are equal to zero. Despite having a trivial path to converge to, there is still an instance of camera retreat. *R*, *P*, and *Y* represent the rotation about the *x, y,* and *z* axis respectively.

The rotational path taken follows an expected convergence path starting at 1 radian and converging to zero, which corresponds to the orientation of the desired camera pose. If you look at the camera position however, something unexpected happens. Despite the motion being purely rotational, there is a small change in depth while trying to reach the desired camera pose. This is a small instance of the camera retreat phenomenon described earlier. Even in trivial cases, the camera still tends to drive itself away from the desired pose. This is the issue this work aims to correct. Camera movements like these create imprecise movements that would not be allowed in an industry with high tolerances. While it is unlikely that the system would go a rotation with such a large rotation, let us consider a smaller rotation that is more likely to occur in practice. Figure 4.3 below shows another rotation about the *z* axis where camera retreat happens in a more extreme fashion.

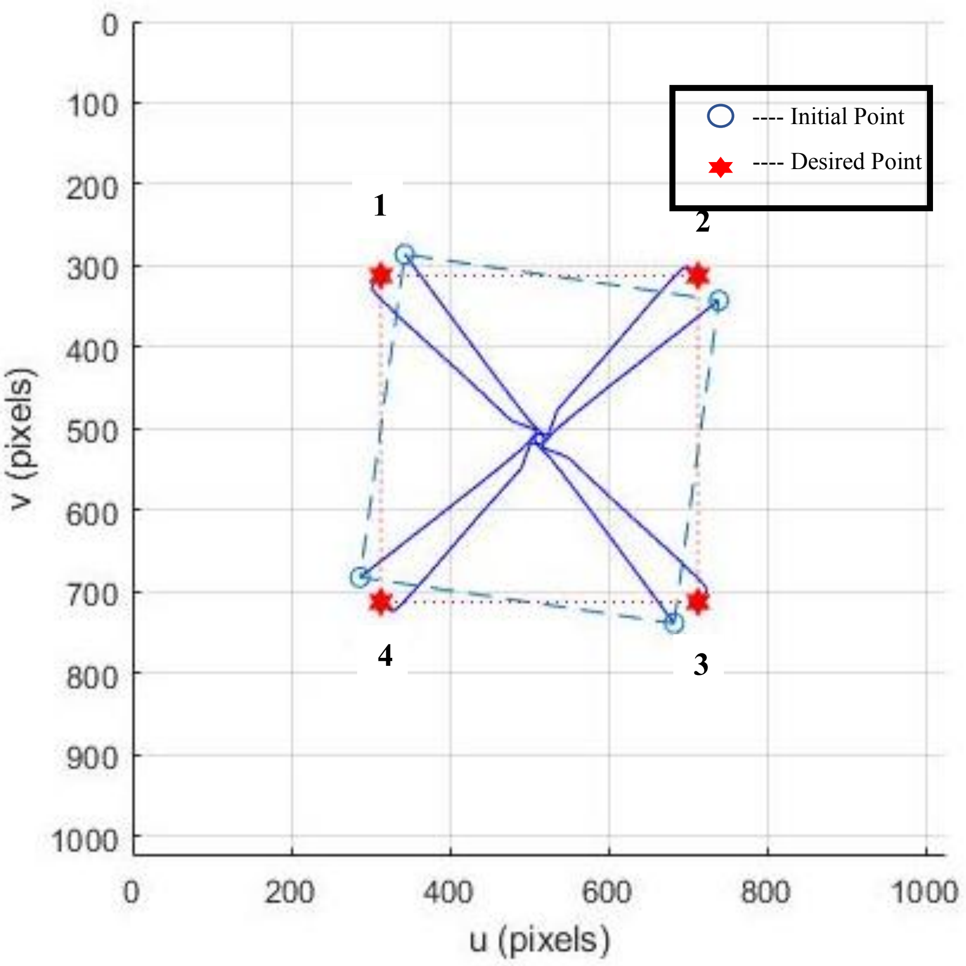


Figure 4.3: Small rotations causing drastic camera retreat problems. The rotation here is 9°.

This slight rotation of 9° is a more likely to happen in practice. Small things that could cause a slight shift in the camera position could cause the camera to drive itself to infinity while trying to servo to the correct location. Figure 4.4 below shows the path the camera took in the *z* direction while attempting to converge.

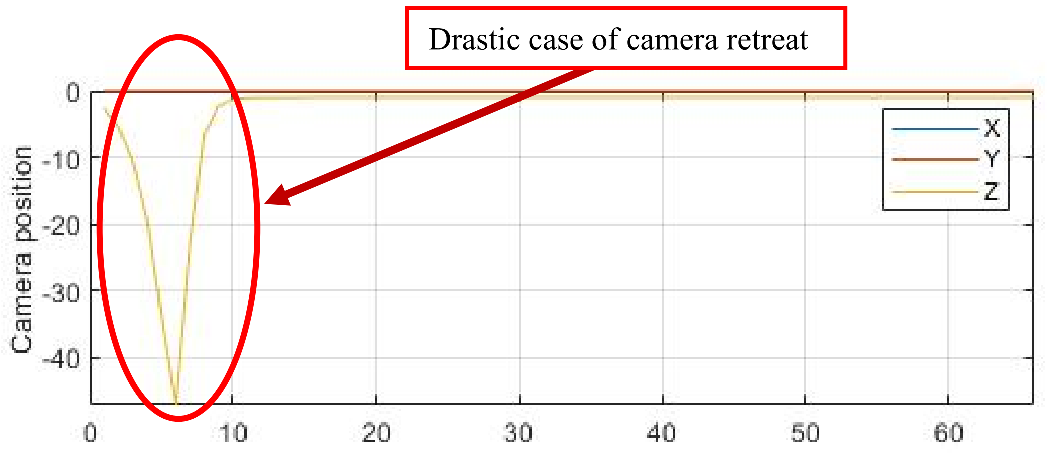


Figure 4.4: Graph of the 9° rotation that caused drastic camera retreat. The depth of the simulation was set to 1 and since the motion was purely rotational, *X* and *Y* are equal to zero. Despite also having a trivial path to converge to, camera retreat is more drastic.

After investigation, it was noticed that camera retreat tends to happen when the rotation is close to some increment of π. Whether the camera rotates at 9° or 171°, it is viewed the same in the IBVS control scheme. Figure 4.5 shows a rotation of 171° and the retreat in the *Z* direction is the same.

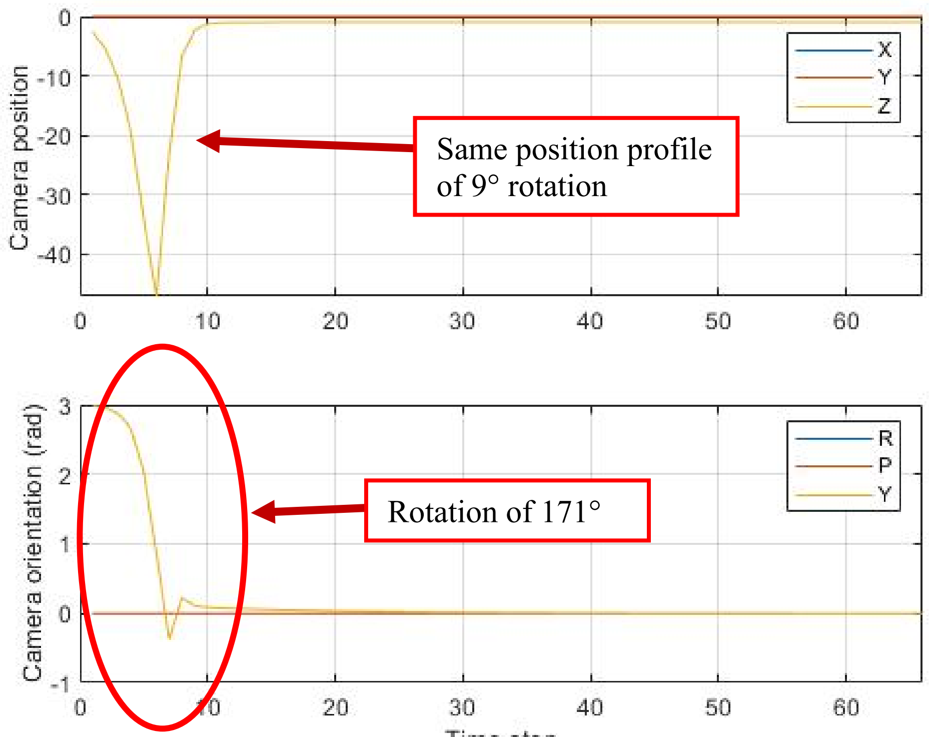


Figure 4.5: Graph of the 171° rotation that also caused drastic camera retreat. The depth of the simulation was set to 1 and since the motion was purely rotational, X and Y are equal to zero. Being that the rotation amount was close to π, camera retreat occurred.

While in the application of detecting flag marker, there will most likely never be a situation where the system would have to perform such a large rotation, it is interesting however to discover that even being in the same range as π can cause the system to perform a suboptimal convergence. Small perturbations in the system like vibrations could cause the camera to follow a drastically different path than necessary. High precision motors and actuators could eradicate this problem completely. For a project like the flag marker detection on wire harnesses for Safran however, the system needed have both high precision and low cost. Low cost motors do not always have smooth and consistent rotations.

The code used to generate this simulation will be listed in Appendix C. In the next section, simulations with DP combined with IBVS will be shown and evaluated. The purpose of these simulations is to show how this partitioned approach address camera retreat and how it can be used to improve IBVS and following an optimal path. We will see that in the same scenario that the system can be driven back to a more optimal path and can be improved by incorporating constraints.

4.3. Adding DP

The most challenging aspect of incorporating DP was developing the algorithm to the traditional IBVS control scheme as shown in section 4.1. As outlined in Chapter 3, the cost function associated with this framework can be partitioned to the desired image space. The code used to develop the simulations in section 4.1 was integrated into the IBVS control scheme used to create the simulations above. The first modification to the code that needed to be made was changing the number of feature points from 3 to 4. This created an 8x6 interaction matrix as opposed to the 6x6 square matrix used in the above simulations. After that, another challenge was dealing with the partitioning of the image space. In the example in section 4.1, the image space was partitioned into a 6 pixel by 6 pixel space. We wanted to keep the same desired image pose in the new simulations, so we used the image space of 1000 pixel by 1000 pixel. The motion that was being examined was so small that having such a large image space required a small number of time steps. The smaller the time steps, the more cost function calculations that must be calculated due to having more options to consider for optimality. The beauty of DP is that we can incorporate constraints into the algorithm without having to modify our initial cost function (Equation (24)). This can be done by taking advantage of the fact that the motion in this example is purely rotational. We know that IBVS will always converge and looking at Figure 4.2 and Figure 4.5, we see that when camera retreat happens the system is still rotating steadily to the correct orientation. This means our cost function only must penalize and update depth. We do not need base the cost on each of the feature points as their coordinates are determined based on the rotation of the square, eliminating the issue we originally had when running the algorithm of having too many options due to calculating the cost at each point for each time step. At each time step, if the depth value is too large, we can send the cost to infinity, eliminating that path as an option and moving on to the next potential option optimality. Figure 4.6 shows the initial results of attempting this combined control scheme on the initial and desired camera pose used in Figure 4.3. In order to penalize depth, it was necessary to drive create constraints for which the depth could not cross. Initially, a limit of -20 was set as an arbitrary value. This can also be seen in Figure 4.6.

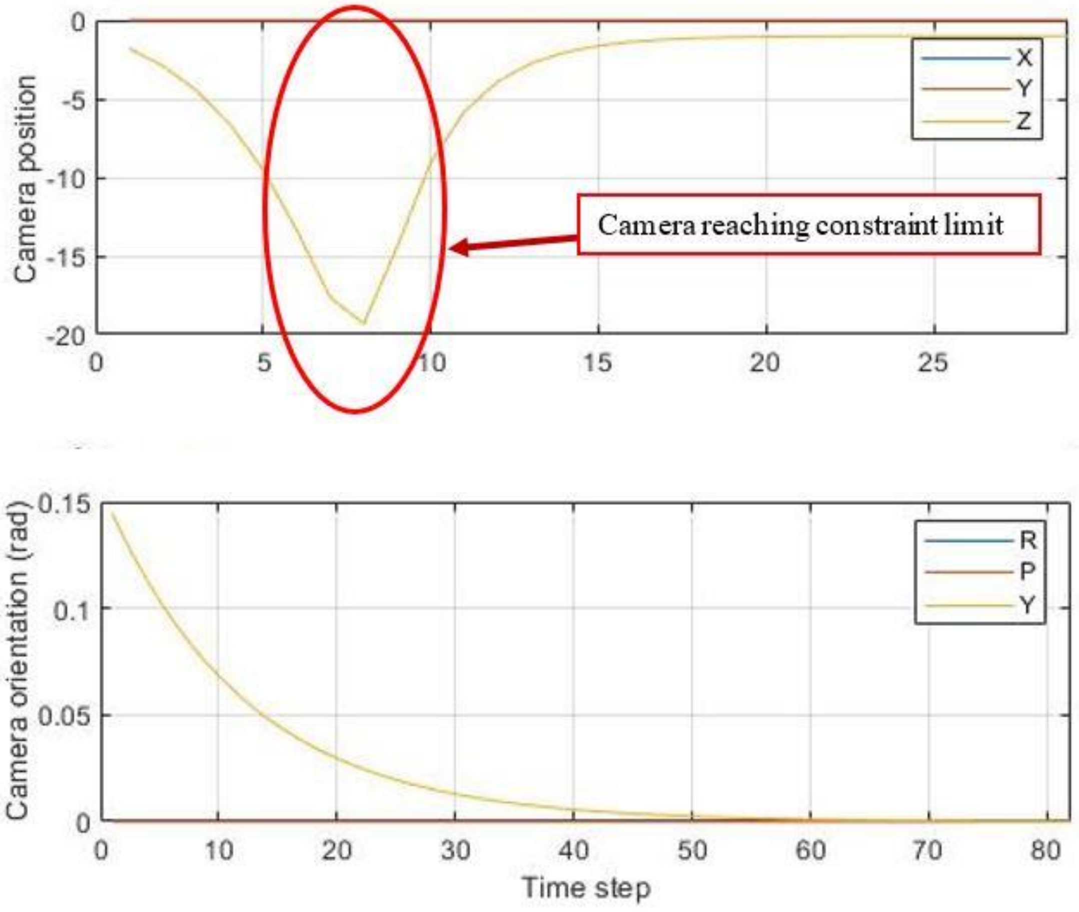


Figure 4.6: Graph of first successful simulation of combined IBVS and DP simulation. Constraints were put on the depth to penalize any attempts to retreat while converging. Rotational convergence followed an expected path.

Something to note with this combined scheme is that the smoothness of the curve is lost. This is due to the constraints in the simulation and detecting the system wanted to approach infinity. Something that is noteworthy is that it did not reach the specific constraint value before turning back. This means that a minimum cost had been determined before reaching the limit of the constraint, thus creating a more optimized path to the desired pose.

Addressing the shape of the curve, we can smooth this by limiting the constraints more. The algorithm is not waiting to reach the constraints but minimizing as the system approaches those constraint values. Figure 4.7 below shows limitation on depth of -4.

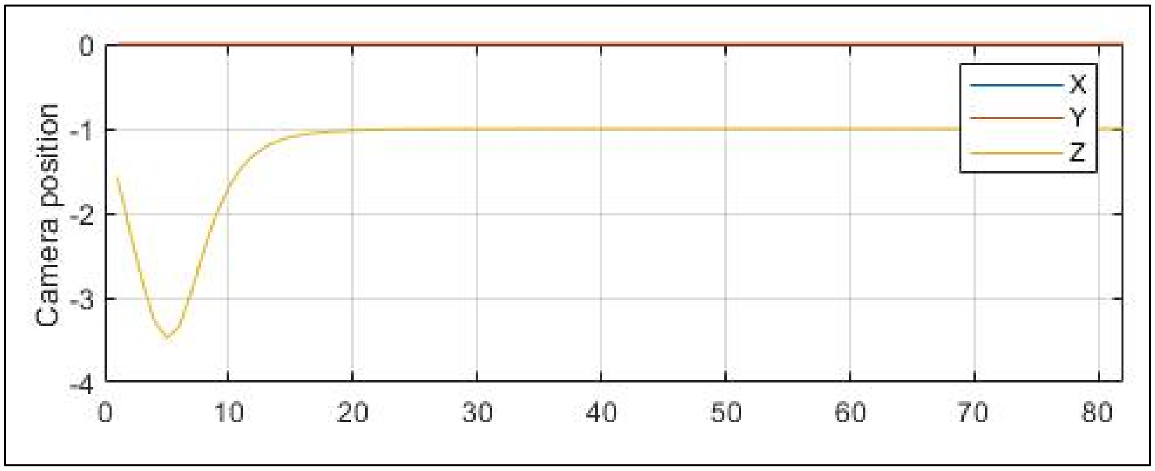


Figure 4.7: Graph of first successful simulation of combined IBVS and DP simulation with depth constraint of -4. Constraints were put on the depth to penalize any attempts to retreat while converging. Rotational convergence followed an expected path.

It can be seen that the curve is a bit smoother and fixes the retreat quicker, however there is still a sharp turn around when fixing the retreat that indicates the minimization is happening to fast. For a real system, this sharp change in motion would cause a motion that is unsmooth and could potential damage the system. We want the constrains within limits that follow a smooth trajectory. By constraining the depth down to -2, we can see in Figure 4.9 a depth that is smooth and more desirable for a real system.

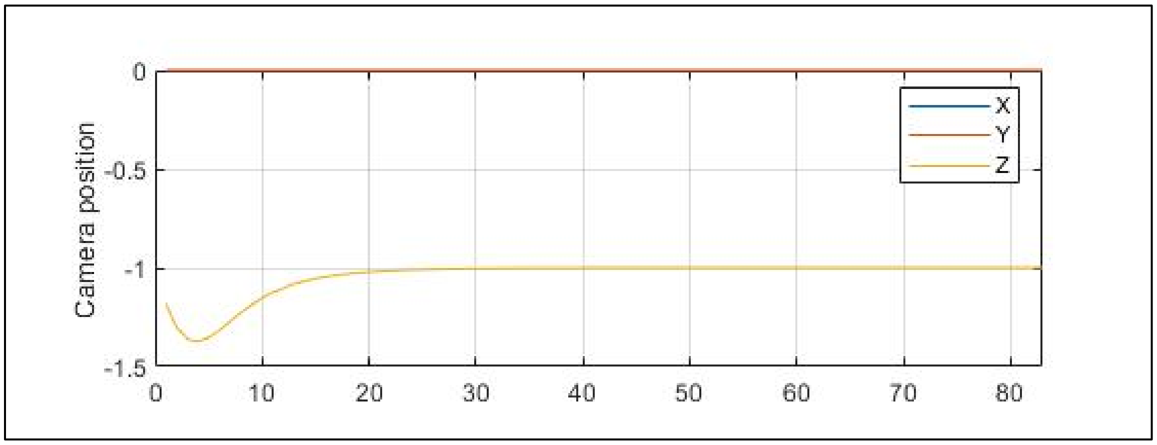


Figure 4.8: Graph of first successful simulation of combined IBVS and DP simulation with depth constraint of -2. Constraints were put on the depth to penalize any attempts to retreat while converging. Rotational convergence followed an expected path.

Here we can see a much smoother control of the depth, significantly minimizing the effects of camera retreat during these specific cases. We can also assume that the trajectory is optimal in position, as optimizing time directly optimizes position.

Lastly, we want to evaluate the performance of scenario that is likely to occur in the process of airplane wire harness inspection; that is a motion that is both translating with the potential to have a slight rotation. As mentioned before, the slightest offset in angle could cause the camera to retreat to infinity or follow a nonoptimal path. We consider the same desired camera pose as the previous examples with a rotation of 9°, however we will translate the object 200 pixels to the left. Figure 4.9 shows the path taken to reach the desired pose.

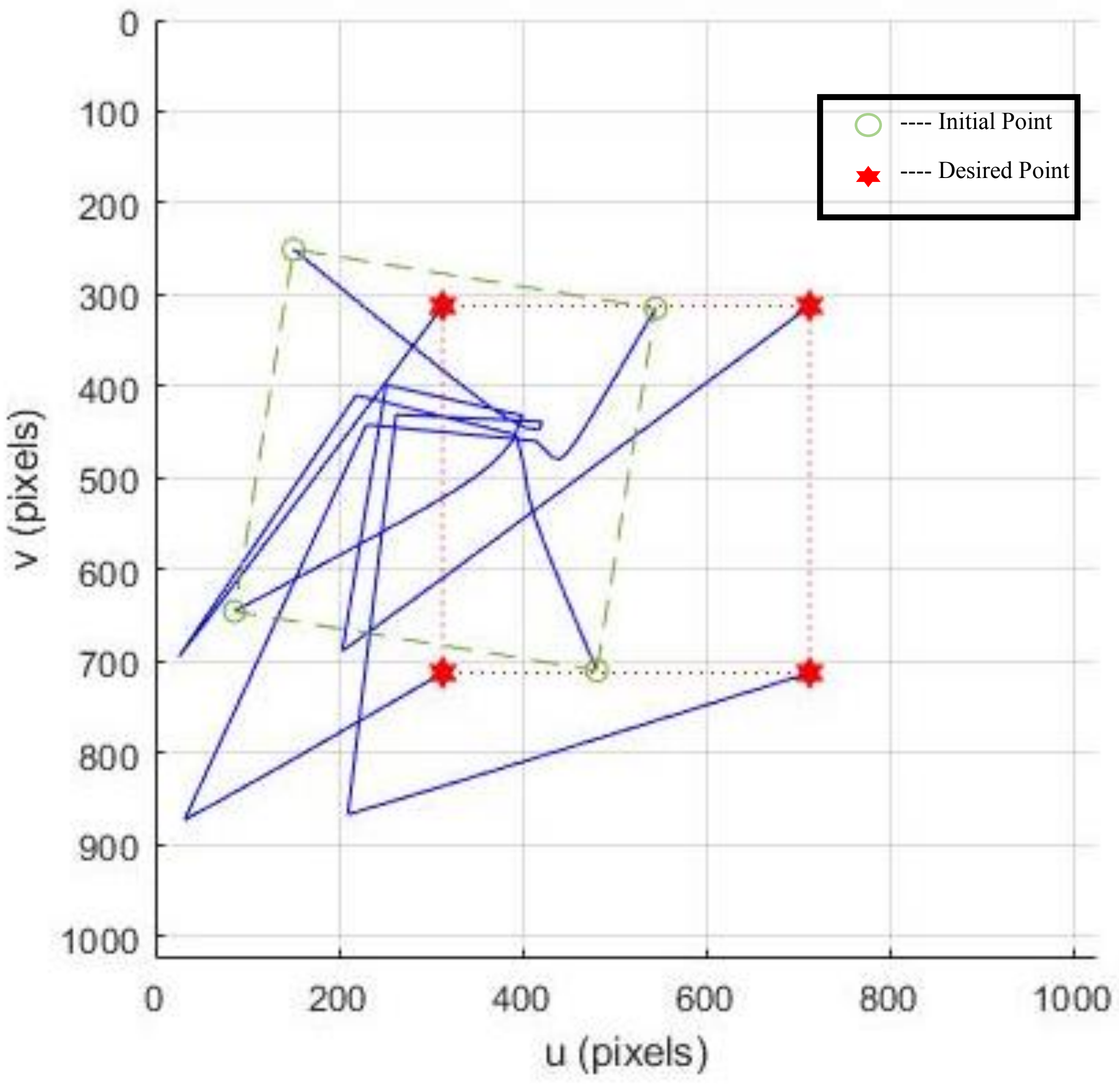


Figure 4.9: Graph of motion of translation and rotation onto a desired camera pose. There were cases of camera retreat in this instance as well, however the path taken to reach the desired camera pose is also not optimal.

Here we notice that there are both instances of camera retreat and following nonoptimal paths. As mentioned in section 2.2, this is due to poor depth estimation. Figure 4.10 shows how the camera converged when trying to reach this state. In this figure we see that when the camera tries to retreat, it also affects the convergence in both the *X* and *Y* directions.

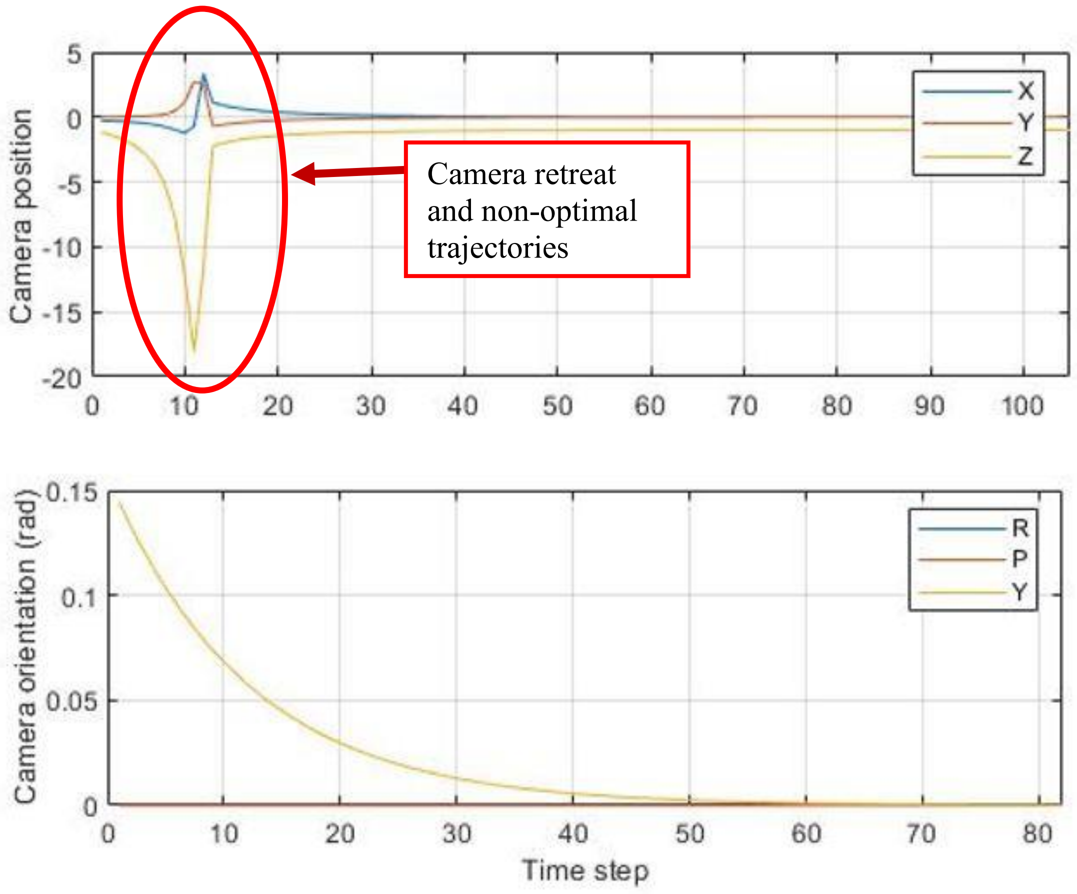


Figure 4.10: A translation with a small rotation causing camera retreat and non-optimal trajectories. The rotation aspect of the trajectory was smooth, again indicating that when using DP, we only must penalize the depth.

It is important to note that this is a niche case. Further investigation would need to be done in order to illustrate why the trajectory followed this specific path, however that is not the focus of the work. Another interesting aspect of this simulation to look at is the cartesian velocities as the simulation was run. For most cases, even the cases shown is Figure 4.3, the velocities follow a smooth curve. This means that despite following a nonoptimal path, the velocity required to reach the desired camera pose was smooth. Figure 4.11 shows these cartesian velocities.

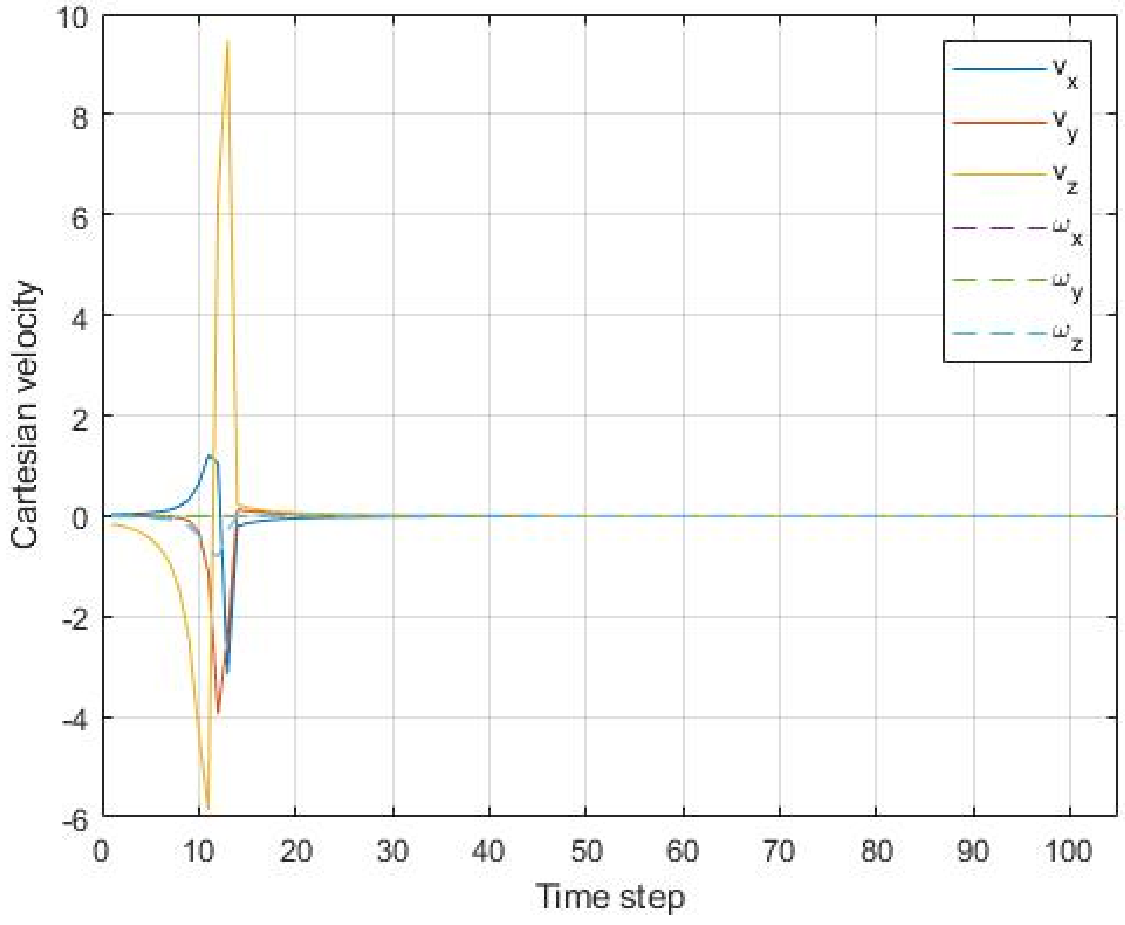


Figure 4.11: Cartesian velocities of translation and rotation simulation. Cartesian velocities indicate that the camera aggressively tried to reach these states.

If we look at the graph, we see spikes in the velocity in all directions, however in the *z* direction there are considerably large spikes in the velocity. It is not known why in this specific case the camera retreat happens significantly faster than in a strictly rotational case, however in the other cases of camera retreat, the velocity follows a much steadier rate. Further investigation would need to be done to better understand why this happens, however for now DP can get the trajectory back on track to following the optimal and more obvious solution. Figure 4.12 shows the same problem with constraints to limit the camera from following this trajectory. We can see in this Figure that DP simply pushed the trajectory on track by monitoring depth. Being that the rotation tends to happen as normal, penalizing the *z* direction from retreating results in a smooth path onto the desired camera pose. The resulting camera position and cartesian velocities can be seen in Figure 4.13 and 35 respectively.

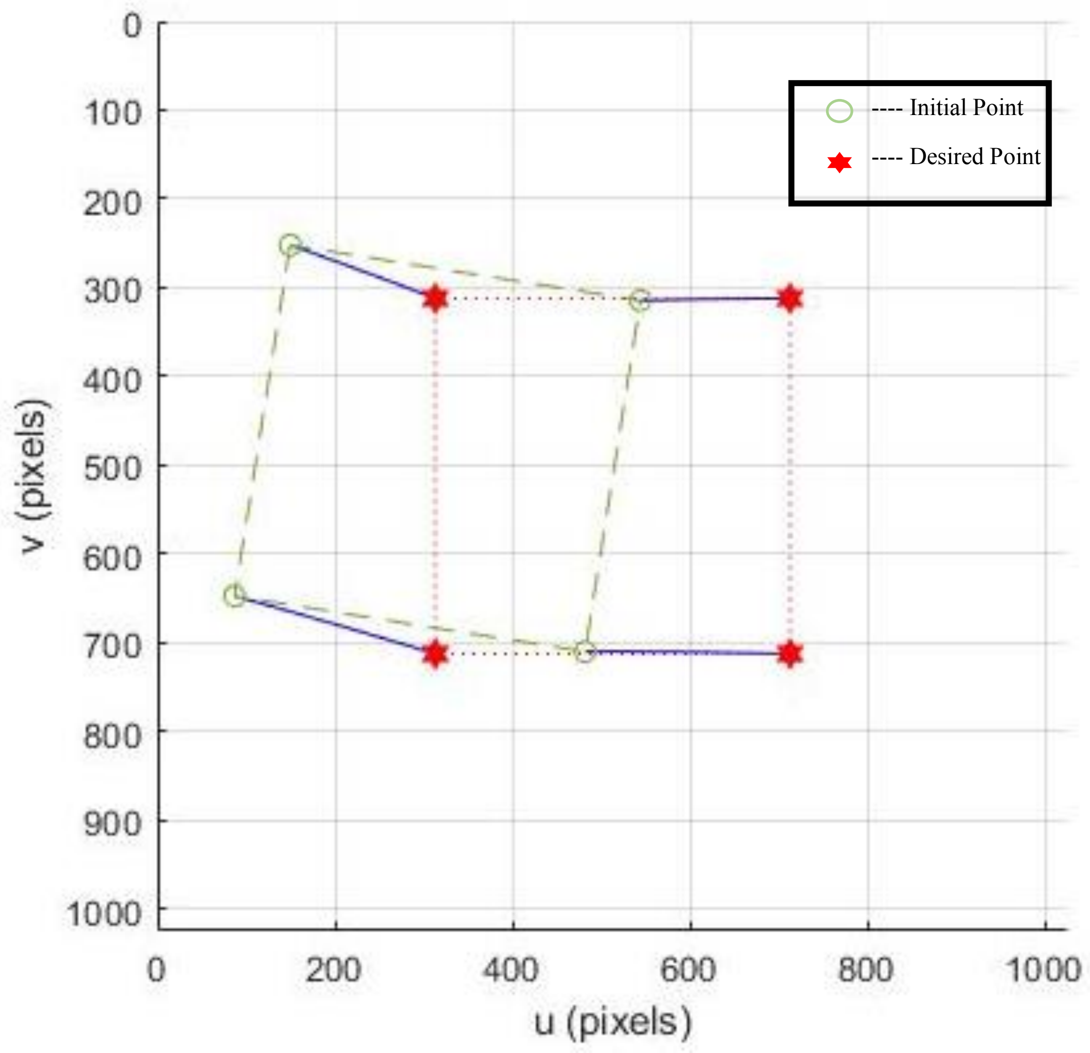


Figure 4.12: Translational and rotational problem with DP, resulting in optimal trajectory in a trivial case.

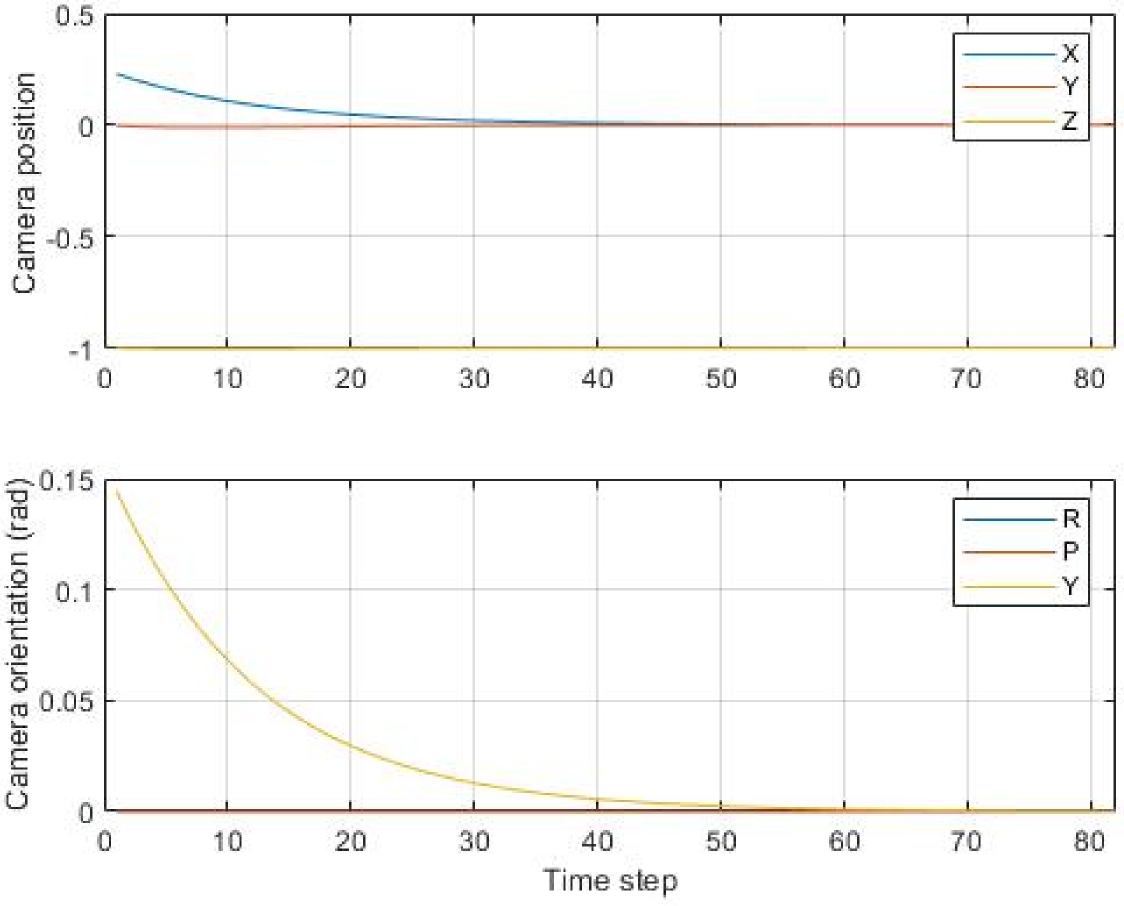


Figure 4.13: Camera position and orientation with DP applied. Depth was constrained to -1.5 which resulted in a smooth trajectory to the desired camera pose.

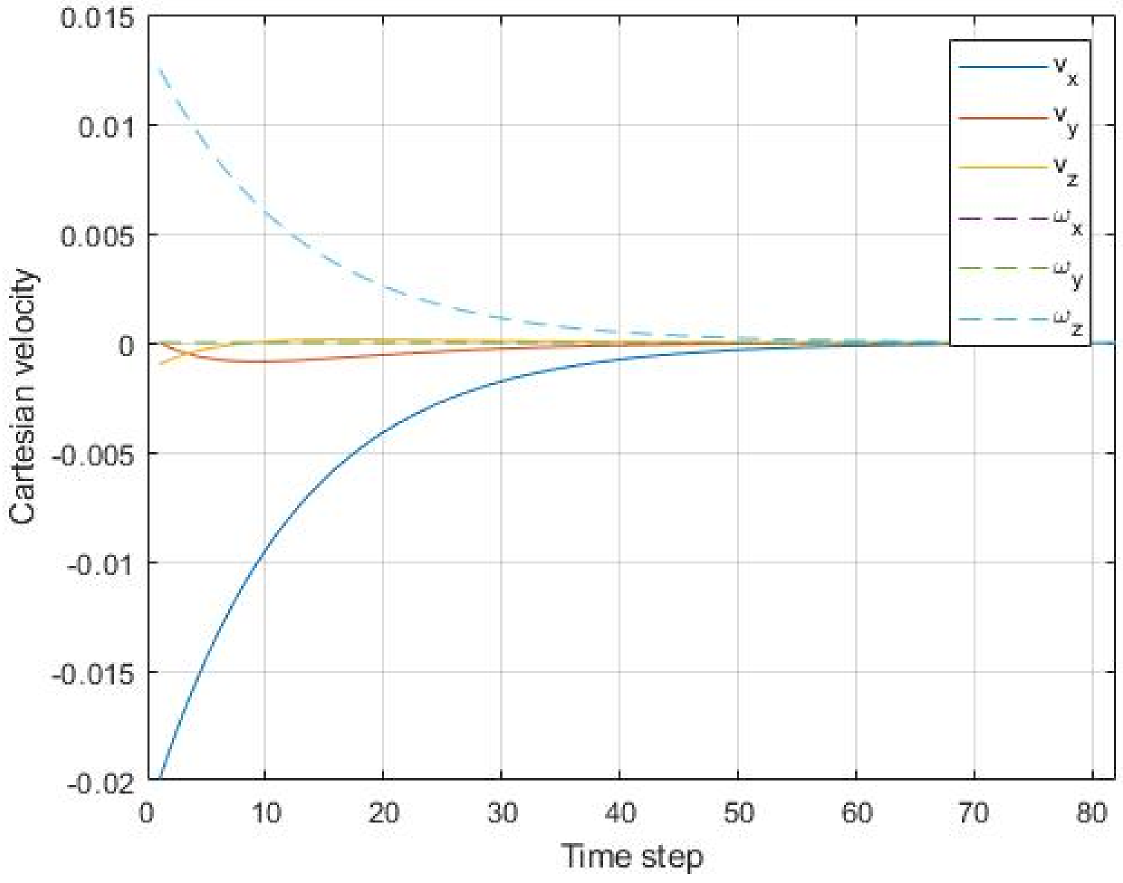


Figure 4.14: Cartesian velocities of translational and rotational problem with DP, resulting in optimal trajectory in a trivial case.

The solution in this scenario shows what the obvious trajectory should be in this scenario. Again, looking at the cartesian velocities, we can see a smooth transition to the desired camera pose.

4.4. Summary

In this chapter, the initial attempts on using a combined IBVS and DP control scheme were explored. One of the drawbacks to using DP is the number of computations required in situations where cost functions for multiple states must be calculated. As we saw, when initially attempting to calculate the cost for 3 feature points, the number of calculations required made it difficult for the algorithm to decide which path was optimal. This could be due to underlying issues with the code, however solving this issue only matters in the case of pure translation. When focusing on the rotational motion only and using that to calculate the cost, we could assess the depth at each state of the rotation to penalize a non-optimal trajectory. This resulted in significantly less cost calculations and a more robust combined control scheme overall. In the specific case of small rotation about the *z* axis, camera retreat appears more likely to occur. A simulation illustrating this was shown after and applying the combined control scheme, the degree at which camera retreat occurred was lowered. It can be improved more by tightening constraints on the depth to get a smoother trajectory to the desired camera pose. Finally, a simulation was conducted where the trajectory that an inspection system examining airplane wire harnesses might follow was run. This simulation included a translation motion with a slight rotation, something that can happen with the camera feed in a low-cost system. In this situation, it seems as if the translation of the camera exaggerated the camera retreat and caused the camera to follow an unorthodox trajectory. The combined control scheme still was able to determine the optimal and trivial path to reach the desired camera pose.

CHAPTER FIVE

Conclusions and Future Work

A combined Image Based Visual Servoing (IBVS) and Dynamic Programming (DP) control scheme was developed and simulated to address the issue of camera retreat that occurs in the traditional IBVS control scheme. This simulation was a partitioned approach to improving IBVS that was also optimal in time. Optimizing in time in turn optimized the distance taken to reach the desired camera pose. In this chapter, we will discuss the results of these simulations, explain the future work and challenges that come with implementing this algorithm onto a real system, and lastly some concluding remarks about high precision automated systems.

5.1. Discussion

When initially attempting to develop the combined control scheme, a key challenge with DP in regards to dimensionality was quickly realized. DP works well for systems of first order as seen in the examples in Chapter 2. The number of states partitioned in this case doesn't drastically increase the number of calculations that need to be stored as. In the example in this work however, the number of states required to be stored is multipled by 3, turning the 1st order system into a 3rd order system. When intially attempting to implement, this wasn't forseen to be a problem, however when simulating this example the number of calculations became too much to store after 1 time step, even after reducing the partitioned space to a 12x12 space. There are a few ways to get around this, however none of them are practical for this application. The first is to simply increase computational storage capabilities. Doing so however, increases the cost and in our case, defeats the purpose of finding a low cost solution to solve the problem. Another method could lie in the development of the code used. The code developed is a brute force method of running the algorithm, however in may not be optimal in computational storage capcity. It may be possible to improve performace by implementing some sort of calculation tracker which eliminates values that are clearly not optimal for determining the desired trajectory, however two issues still arise from this. It will still be necessary to calculate those "unneccesary values" which still takes time and doesn't significantly decrease computation time and second, it reduces the robustness in a sense that those cost values would need to be recalculated everytime a different orientation is desired. Once the values are stored, it can be used for different trajectories as long as the partitioned image space is the same. The last method that could decrease computational storage capacity is a method known as "incremental dynamic programming" [43]. The premise of this method takes the traditional dynamic programming scheme and targets specific state values for each state variable instead of considering the cost for the entire space for each variable. It works well for higher order systems as the cost value for one state variables determines the options the other state variables can take within a limited range [44][45]. Formulating the control scheme this way would be another way to implement DP with IBVS, however as we saw determining the cost based on the rotation of the camera is a better state variable to penalize as it allows for a large reduction in calculations. Applying incremental dynamic programming in theory would still require more calculations than the approach taken in this work, however it would require significantly less calculations than the traditional DP control scheme. The initial simulations gave insight into how determining which state variable to choose for calculating cost.

Once the correct variable was determined, an IBVS simulation was run and compared to what was expected of the traditional scheme. As mentioned before, IBVS struggles with small rotations around the *z* axis. Specifically the closer it is to multiples of π. We can see that despite having a smooth convergence in towards the desired camera pose, there was still a retreat in the *z* direction through a phenomenon known as camera retreat. In this scenario, the degree at which the retreat happened is small, however this performance was expected from the traditional scheme in this scenario. Another case of extreme camera retreat was illustrated with a smaller rotation about the *z* axis and the camera retreated in an even more dramatic fashion. Again, the performance in this scenario was expected and validated that the open source code we used to develop the initial simulation was behaving as predicited. At this stage, this code was modified to impliment DP using the algorithms developed in Chapter 2 and the code used to create the initial combined simulations in Chapter 4. After incorporating DP with constraints, the degree at which camera retreat happened began to diminish. The initial constraints that were set caused the motion of the camera to not be smooth however it drastically limited the degree at which camera retreat happened. This jaggedness is caused from the initial constraints set on *z* axis. As the system began to approach this constraint, the algorithm began correct the retreat faster in order to not reach the constraint. While this fixed the camera retreat problem, a jagged response like this would not be ideal in application. In order to improve the performance, tighter constraints were implemented. Limiting the amount the camera could retreat in the *z* direction drastically decreased the issue of camera retreat in this specific scenario, but also smoothed the trajectory of the convergence. Notice however that the algorithm does not completely remove camera retreat. Setting the constraint in the *z* direction to the exact desired depth in this scenario causes the simulation to fail. This is because the simulation is purely rotational and the traditional IBVS control scheme needs to follow this trajectory in order to complete the motion. Once the algorithm notices the non optimal trajectory, it begins to drive the system back to the optimal path.

For most applications, it is rare that you have a strictly rotational desired camera pose. A simulation with a motion that is more likely to occurr, including a translation and a rotation of some sort, was conducted. In a combined translational and rotational control scenario however, we notice the issue of camera retreat still presents itself in the rotational aspect of the trajectory. We also notice that the trajectory not only retreats, but follows an nonoptimal trajectory. As mentioned before, a small rotation is likely to occurr in a low cost system due to vibrations or in attempts to drive the system to the desired camera pose. The scenario in this simulation can happen quite commonly and we noticed this small rotation in the camera frame during the image capturing of our initial physical prototype. As seen in the results of the simulation, the combined control scheme worked excceptionally well, completely smoothing the trajectory and following the trivial trajectory as one one predict. The improvement of the performance from implementing the algorithm is believed to stem from how extreme the trajectory was initially supposed to be. There could be investigation into the specifics of this scenario, but rather than try to understand why the simulation followed this specific trajectory, it is more useful to simply correct it, as it is niche case. Small rotations have been known to follow unusual trajectories, and the speed at which they converge allows DP to accurately and effectively correct it. Based on the results, in situations where high precision and accurate detection are required, this combined control scheme allows IBVS to be implemented without camera retreat interfering with the convergence.

5.2. Future work

In conclusion, Image Based Visual Servoing (IBVS) has limitations when it comes to converging to the desired camera pose optimally. When converging along a trajectory where a small rotation is involved, it can either retreat far away from the desired camera pose through a phenomenon known as camera retreat, or it can follow a suboptimal trajectory in trivial cases. These issues limit the application range for IBVS, specifically in applications that require high precision like the aerospace manufacturing industry. Partitioned approaches have been used to address these shortcomings, however these approaches do not guarantee optimality. Dynamic programing (DP) is a partitioned optimal control technique that can guarantee optimality of a specific control criteria of interest. In this work, DP was applied to IBVS to address the issue of camera retreat while also optimizing the path taken to converge onto the desired camera pose. The implementation of this control technique prevented camera retreat and following suboptimal trajectories while maintaining the beauty of the traditional IBVS control scheme. While this simulation worked well, there are still area that require further investigation, specifically when implementing the algorithm onto a real system.

The first challenge comes with image data. It is required to know what the image coordinates of the desired camera pose are, store that data, and be able to synchronize the image with where the image is in the camera frame. This would require further investigation into the area of image processing. While earlier in this work features of interest were able to be detected, there was no information being gathered about where the image was in space as the feature detection and servoing of the motor worked separately. In the simulation, we can easily characterize where the desired camera pose is as we are not extracting it from a real image. Another challenge with implementing the algorithm onto a real system is choosing the right system for incorporating dynamics. The simulations were based on a system that could move in any direction freely. Future work would be spent exploring these areas to implement on a real system.

APPENDICES

Appendix A
Interaction Matrix Derivation

The Interaction (Jacobian) matrix is a matrix of intrinsic camera properties that relates the velocity of the camera to the velocity of the image coordinates. In this section, the derivation for this matrix will be developed by incorporating concepts of projecting 3D points onto a 2D plane. Combining these concepts with dynamics of a rigid body, we can develop the relationship. Let's take a point in 3D space defined as $r = (x, y, z)$ and its corresponding image plane coordinates in 2D space as $f = (u, v)$. The projection between the 3D and 2D space is given by Equation (A1):

$$\begin{aligned} u &= \frac{x}{z} \\ v &= \frac{y}{z} \end{aligned} \tag{A1}$$

By taking the time derivative of Equation (A1), we get Equation (A2)

$$\begin{aligned} \dot{u} &= \frac{\dot{x}z - x\dot{z}}{z^2} = \frac{\dot{x} - x\dot{z}}{z} \\ \dot{v} &= \frac{\dot{y}z - y\dot{z}}{z^2} = \frac{\dot{y} - y\dot{z}}{z} \end{aligned} \tag{A2}$$

The equations represent the velocity of the image points on the 2D plane. From rigid body dynamics, we can represent the velocity of a body in space is by Equation (A3):

$$\dot{r} = -v_c - \omega_c \times r \leftrightarrow \begin{cases} \dot{x} = -v_x - \omega_y z + \omega_z y \\ \dot{y} = -v_y - \omega_z x + \omega_x z \\ \dot{z} = -v_z - \omega_x y + \omega_y x \end{cases} \tag{A3}$$

Where v_c and ω_c represent the translational and rotational velocity respectively. As stated before, r is the 3D representation of the camera point in space. We can insert the rigid body dynamics from Equation (A3) into Equation (A2) and group the terms to get

$$\begin{aligned}\dot{u} &= -\frac{v_x}{z} + \frac{xv_z}{z} + xy\omega_x - (1 - x^2)\omega_y + y\omega_z \\ \dot{v} &= -\frac{v_y}{z} + \frac{yv_z}{z} + (1 + y^2)\omega_x - xy\omega_y - x\omega_z\end{aligned} \tag{A4}$$

Which can be written in the form

$$\dot{f} = L_s\dot{r} \tag{A5}$$

Where the interaction matrix L_s is

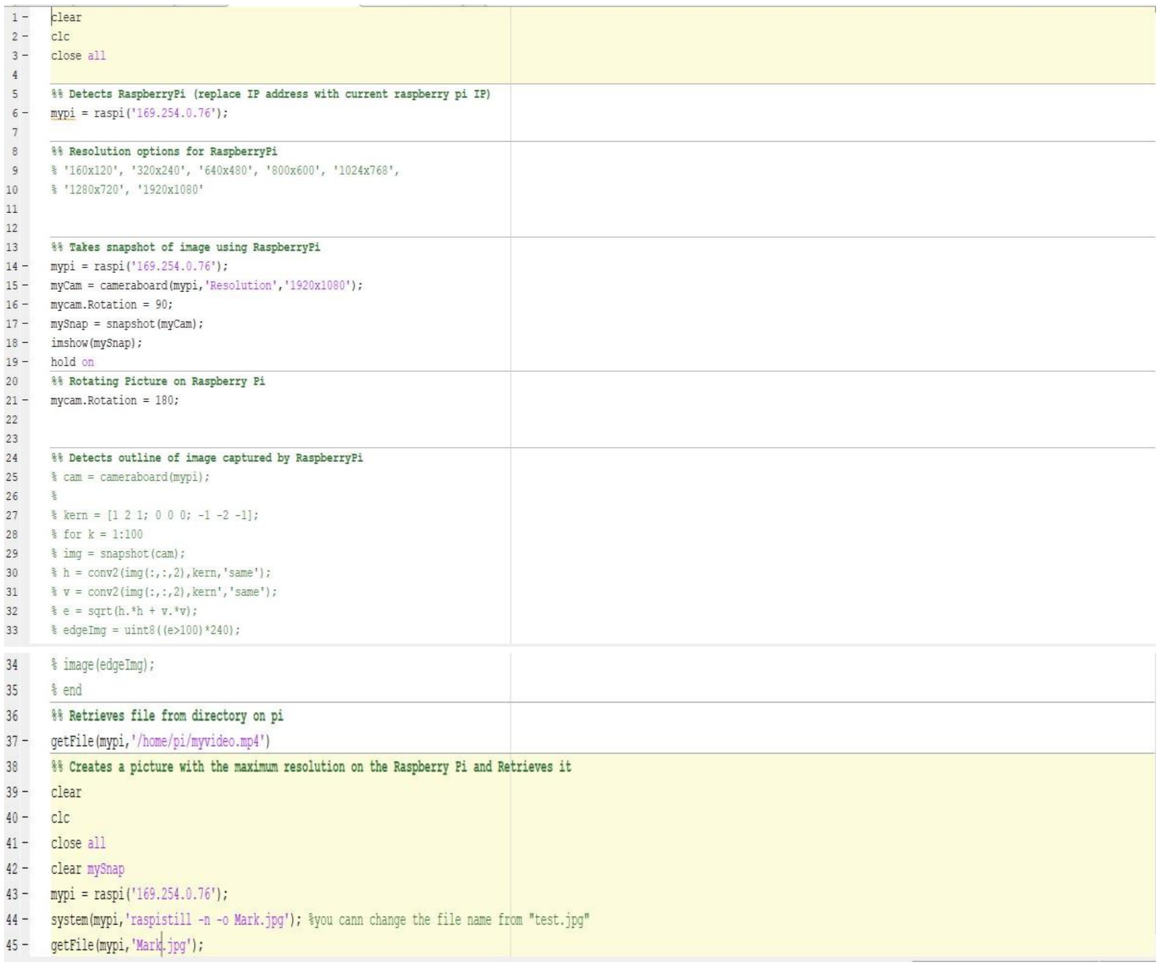
$$L_s = \begin{bmatrix} \frac{1}{z} & 0 & -\frac{u}{z} & -uv & (1^2 + u^2) & -v \\ 0 & \frac{1}{z} & -\frac{v}{z} & -1^2 - v^2 & uv & u \end{bmatrix} \tag{A6}$$

As mentioned in Chapter 2, we can use this relationship to develop a control law for a camera and the image points. Not that this derivation assumes a focal length of one, which is why Equation (A6) differs from Equation (7) [29].

Appendix B
Code for Human detection and Edge detection

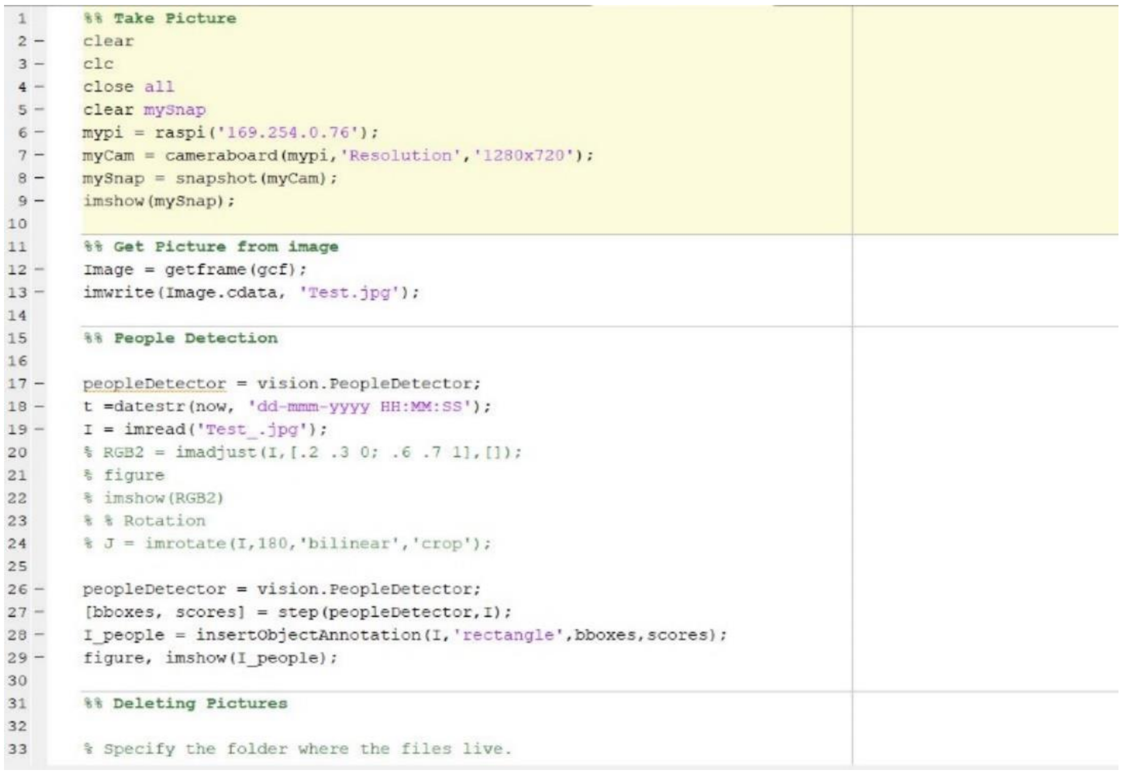
Human Detection (MATLAB)

- Camera Initialization

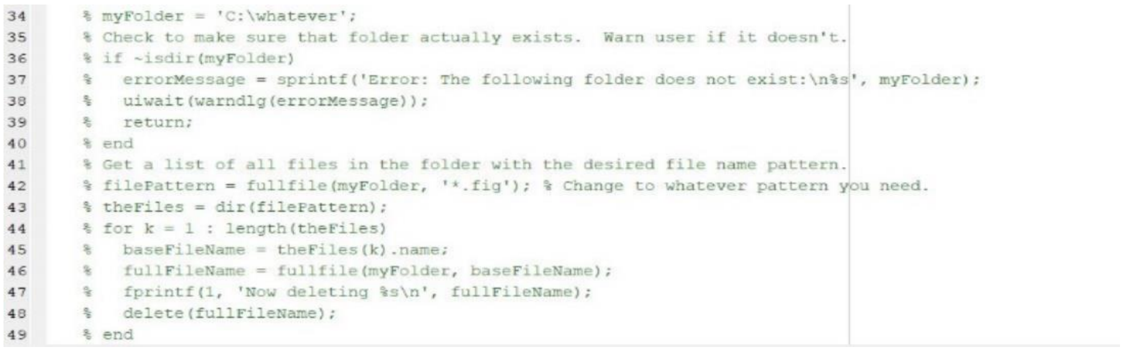


```matlab
clear
clc
close all

%% Detects RaspberryPi (replace IP address with current raspberry pi IP)
mypi = raspi('169.254.0.76');

%% Resolution options for RaspberryPi
% '160x120', '320x240', '640x480', '800x600', '1024x768',
% '1280x720', '1920x1080'


%% Takes snapshot of image using RaspberryPi
mypi = raspi('169.254.0.76');
myCam = cameraboard(mypi,'Resolution','1920x1080');
mycam.Rotation = 90;
mySnap = snapshot(myCam);
imshow(mySnap);
hold on
%% Rotating Picture on Raspberry Pi
mycam.Rotation = 180;


%% Detects outline of image captured by RaspberryPi
% cam = cameraboard(mypi);
%
% kern = [1 2 1; 0 0 0; -1 -2 -1];
% for k = 1:100
% img = snapshot(cam);
% h = conv2(img(:,:,2),kern,'same');
% v = conv2(img(:,:,2),kern','same');
% e = sqrt(h.*h + v.*v);
% edgeImg = uint8((e>100)*240);
% image(edgeImg);
% end
%% Retrieves file from directory on pi
getFile(mypi,'/home/pi/myvideo.mp4')
%% Creates a picture with the maximum resolution on the Raspberry Pi and Retrieves it
clear
clc
close all
clear mySnap
mypi = raspi('169.254.0.76');
system(mypi,'raspistill -n -o Mark.jpg'); %you cann change the file name from "test.jpg"
getFile(mypi,'Mark.jpg');
```

- Detection of human



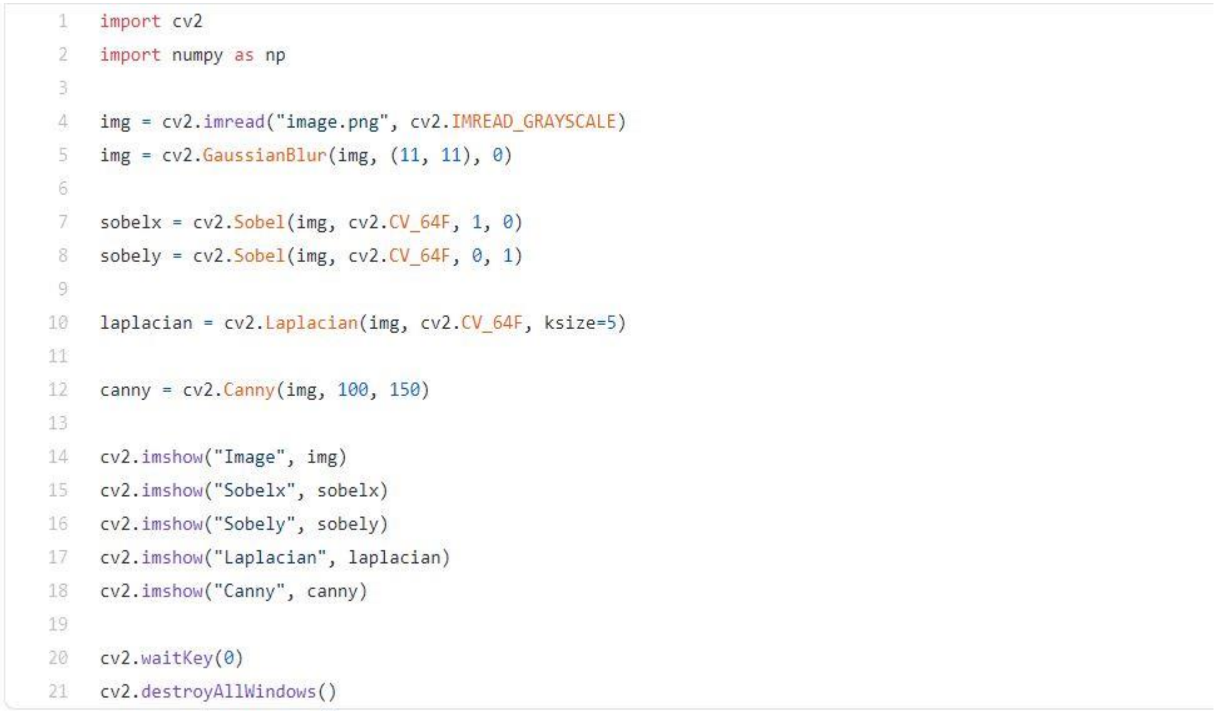
```
%% Take Picture
clear
clc
close all
clear mySnap
mypi = raspi('169.254.0.76');
myCam = cameraboard(mypi,'Resolution','1280x720');
mySnap = snapshot(myCam);
imshow(mySnap);

%% Get Picture from image
Image = getframe(gcf);
imwrite(Image.cdata, 'Test.jpg');

%% People Detection

peopleDetector = vision.PeopleDetector;
t =datestr(now, 'dd-mmm-yyyy HH:MM:SS');
I = imread('Test_.jpg');
% RGB2 = imadjust(I,[.2 .3 0; .6 .7 1],[]);
% figure
% imshow(RGB2)
% % Rotation
% J = imrotate(I,180,'bilinear','crop');

peopleDetector = vision.PeopleDetector;
[bboxes, scores] = step(peopleDetector,I);
I_people = insertObjectAnnotation(I,'rectangle',bboxes,scores);
figure, imshow(I_people);

%% Deleting Pictures

% Specify the folder where the files live.
```



```
% myFolder = 'C:\whatever';
% Check to make sure that folder actually exists.  Warn user if it doesn't.
% if ~isdir(myFolder)
%   errorMessage = sprintf('Error: The following folder does not exist:\n%s', myFolder);
%   uiwait(warndlg(errorMessage));
%   return;
% end
% Get a list of all files in the folder with the desired file name pattern.
% filePattern = fullfile(myFolder, '*.fig'); % Change to whatever pattern you need.
% theFiles = dir(filePattern);
% for k = 1 : length(theFiles)
%   baseFileName = theFiles(k).name;
%   fullFileName = fullfile(myFolder, baseFileName);
%   fprintf(1, 'Now deleting %s\n', fullFileName);
%   delete(fullFileName);
% end
```

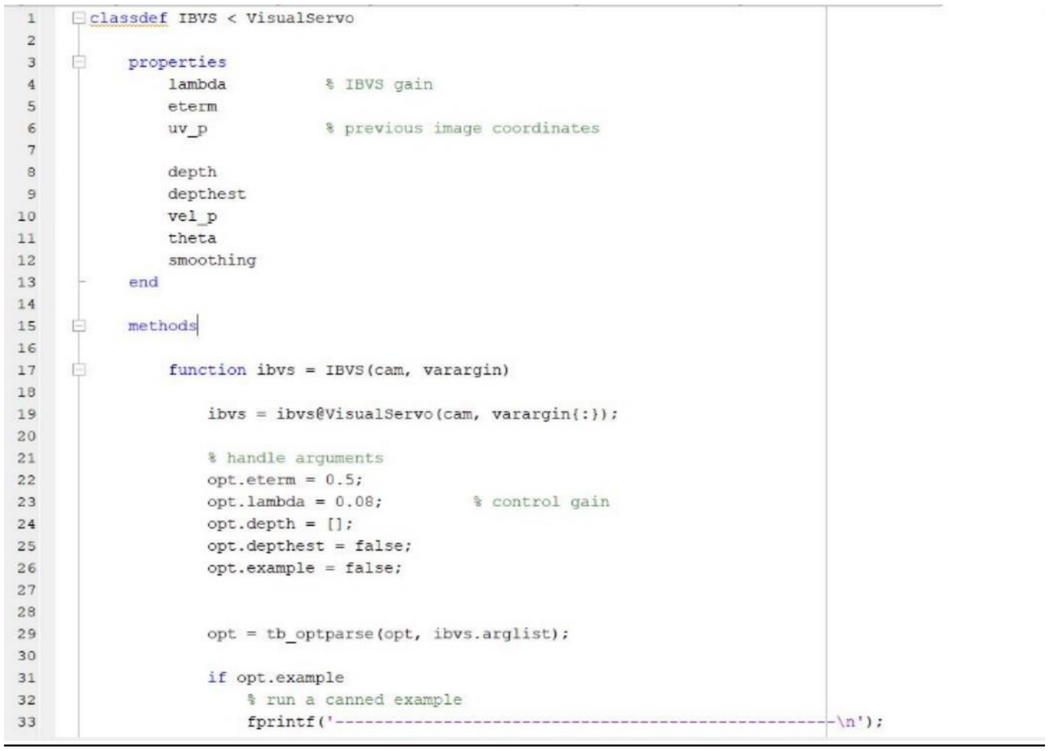
Basic Edge detection

This code is based off open source OpenCV code. This laid the frame work for the simulations.

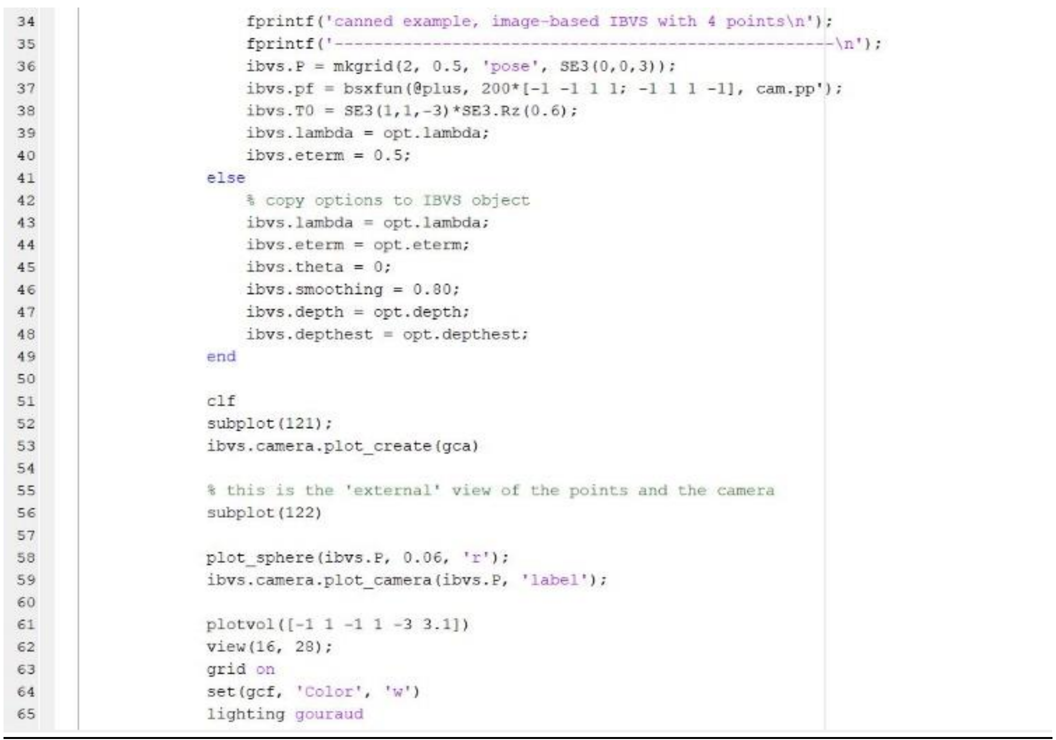


```python
import cv2
import numpy as np

img = cv2.imread("image.png", cv2.IMREAD_GRAYSCALE)
img = cv2.GaussianBlur(img, (11, 11), 0)

sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0)
sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1)

laplacian = cv2.Laplacian(img, cv2.CV_64F, ksize=5)

canny = cv2.Canny(img, 100, 150)

cv2.imshow("Image", img)
cv2.imshow("Sobelx", sobelx)
cv2.imshow("Sobely", sobely)
cv2.imshow("Laplacian", laplacian)
cv2.imshow("Canny", canny)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

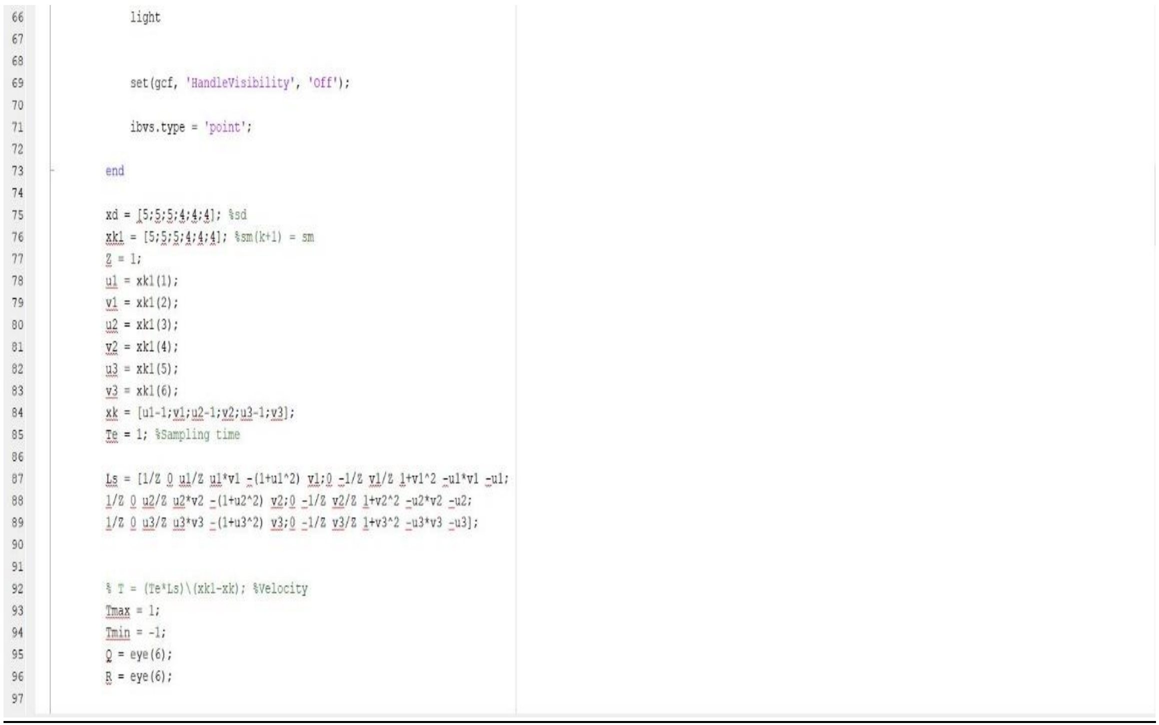
<u>Appendix C</u>
<u>Combined IBVS Control Scheme</u>



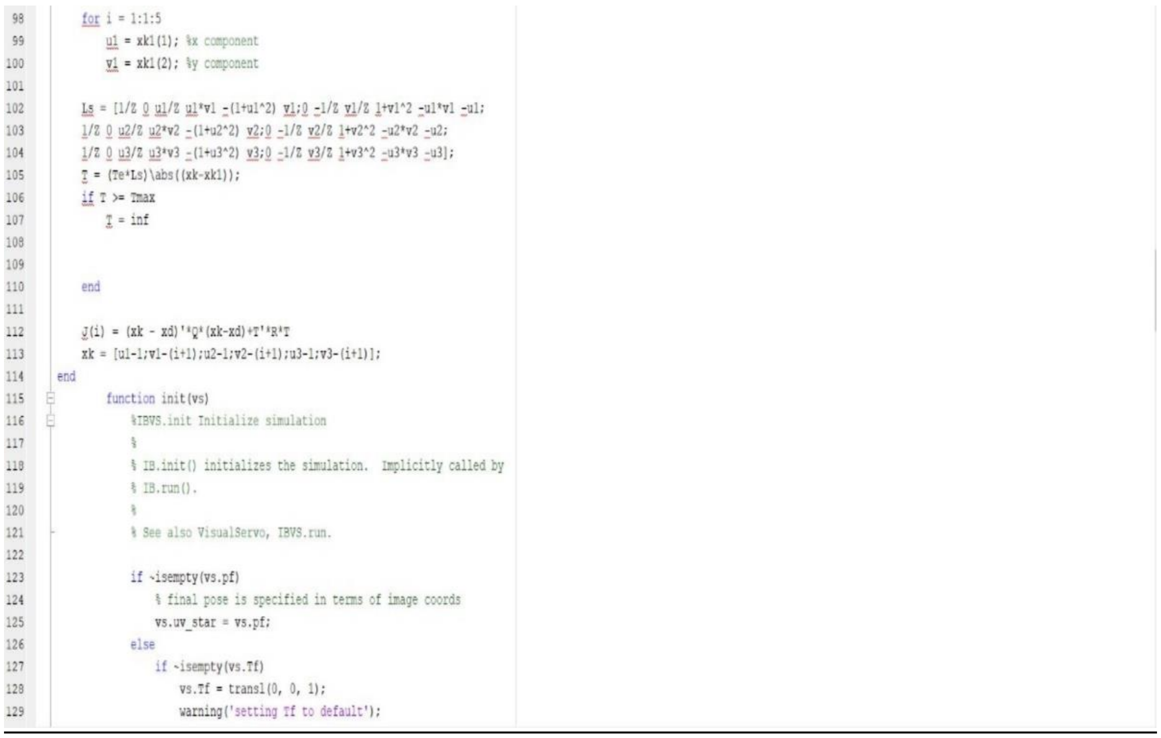
```
classdef IBVS < VisualServo

    properties
        lambda          % IBVS gain
        eterm
        uv_p            % previous image coordinates

        depth
        depthest
        vel_p
        theta
        smoothing
    end

    methods

        function ibvs = IBVS(cam, varargin)

            ibvs = ibvs@VisualServo(cam, varargin{:});

            % handle arguments
            opt.eterm = 0.5;
            opt.lambda = 0.08;          % control gain
            opt.depth = [];
            opt.depthest = false;
            opt.example = false;


            opt = tb_optparse(opt, ibvs.arglist);

            if opt.example
                % run a canned example
                fprintf('---------------------------------------------------\n');
```



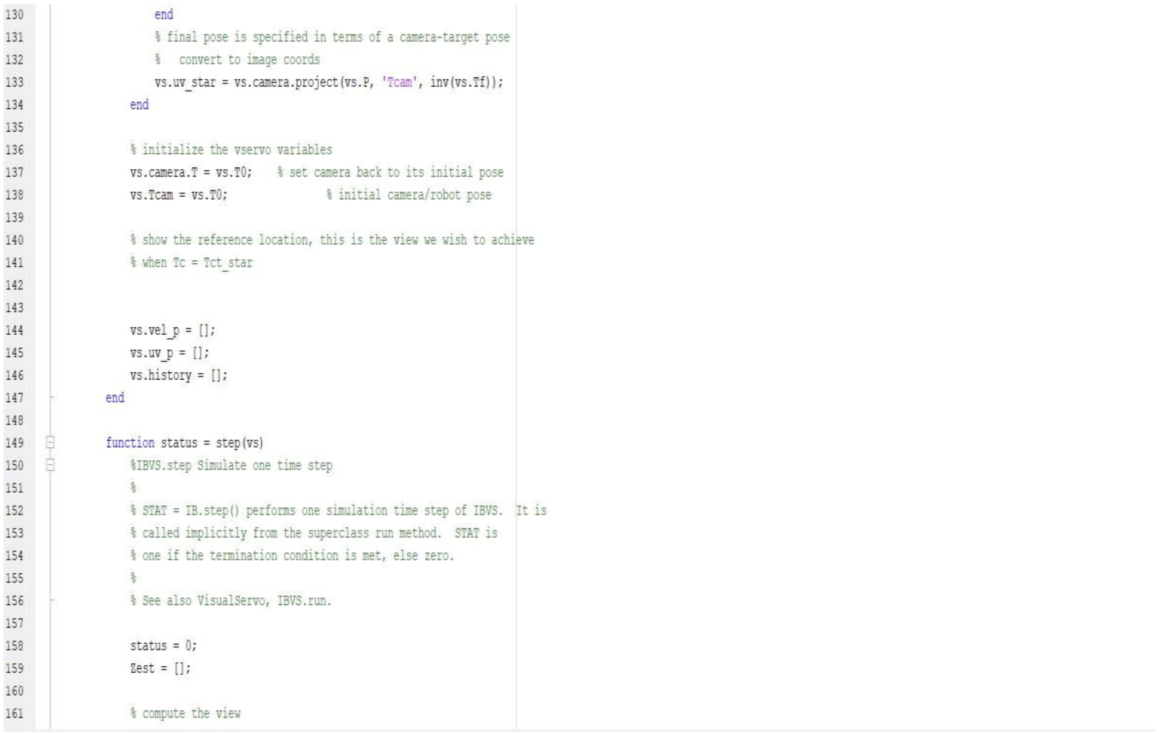
```
                fprintf('canned example, image-based IBVS with 4 points\n');
                fprintf('---------------------------------------------------\n');
                ibvs.P = mkgrid(2, 0.5, 'pose', SE3(0,0,3));
                ibvs.pf = bsxfun(@plus, 200*[-1 -1 1 1; -1 1 1 -1], cam.pp');
                ibvs.T0 = SE3(1,1,-3)*SE3.Rz(0.6);
                ibvs.lambda = opt.lambda;
                ibvs.eterm = 0.5;
            else
                % copy options to IBVS object
                ibvs.lambda = opt.lambda;
                ibvs.eterm = opt.eterm;
                ibvs.theta = 0;
                ibvs.smoothing = 0.80;
                ibvs.depth = opt.depth;
                ibvs.depthest = opt.depthest;
            end

            clf
            subplot(121);
            ibvs.camera.plot_create(gca)

            % this is the 'external' view of the points and the camera
            subplot(122)

            plot_sphere(ibvs.P, 0.06, 'r');
            ibvs.camera.plot_camera(ibvs.P, 'label');

            plotvol([-1 1 -1 1 -3 3.1])
            view(16, 28);
            grid on
            set(gcf, 'Color', 'w')
            lighting gouraud
```



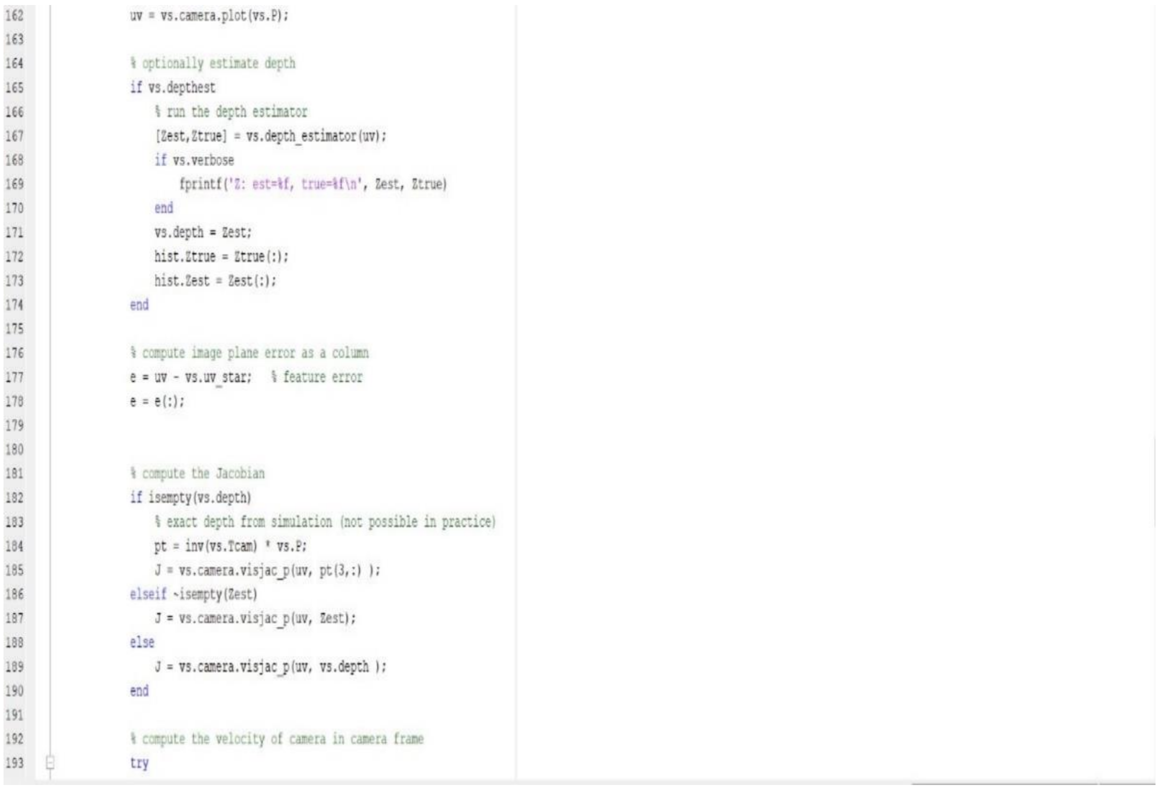
```
            light


            set(gcf, 'HandleVisibility', 'Off');

            ibvs.type = 'point';

        end

        xd = [5;5;5;4;4;4]; %sd
        xk1 = [5;5;5;4;4;4]; %sm(k+1) = sm
        Z = 1;
        u1 = xk1(1);
        v1 = xk1(2);
        u2 = xk1(3);
        v2 = xk1(4);
        u3 = xk1(5);
        v3 = xk1(6);
        xk = [u1-1;v1;u2-1;v2;u3-1;v3];
        Te = 1; %Sampling time

        Ls = [1/Z 0 u1/Z u1*v1 -(1+u1^2) v1;0 -1/Z v1/Z 1+v1^2 -u1*v1 -u1;
        1/Z 0 u2/Z u2*v2 -(1+u2^2) v2;0 -1/Z v2/Z 1+v2^2 -u2*v2 -u2;
        1/Z 0 u3/Z u3*v3 -(1+u3^2) v3;0 -1/Z v3/Z 1+v3^2 -u3*v3 -u3];


        % T = (Te*Ls)\(xk1-xk); %Velocity
        Tmax = 1;
        Tmin = -1;
        Q = eye(6);
        R = eye(6);

```



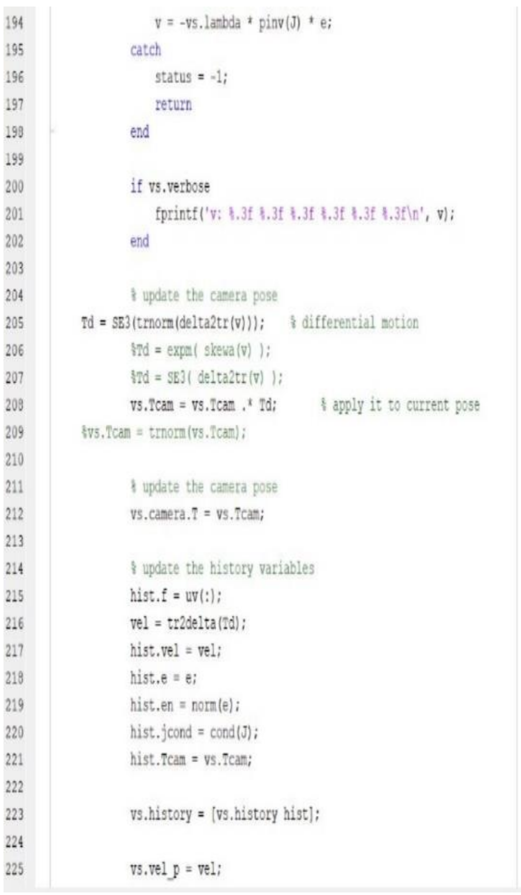
```
    for i = 1:1:5
        u1 = xk1(1); %x component
        v1 = xk1(2); %y component

    Ls = [1/Z 0 u1/Z u1*v1 -(1+u1^2) v1;0 -1/Z v1/Z 1+v1^2 -u1*v1 -u1;
    1/Z 0 u2/Z u2*v2 -(1+u2^2) v2;0 -1/Z v2/Z 1+v2^2 -u2*v2 -u2;
    1/Z 0 u3/Z u3*v3 -(1+u3^2) v3;0 -1/Z v3/Z 1+v3^2 -u3*v3 -u3];
    T = (Te*Ls)\abs((xk-xk1));
    if T >= Tmax
        T = inf


    end

    J(i) = (xk - xd)'*Q*(xk-xd)+T'*R*T
    xk = [u1-1;v1-(i+1);u2-1;v2-(i+1);u3-1;v3-(i+1)];
end
        function init(vs)
            %IBVS.init Initialize simulation
            %
            % IB.init() initializes the simulation.  Implicitly called by
            % IB.run().
            %
            % See also VisualServo, IBVS.run.

            if ~isempty(vs.pf)
                % final pose is specified in terms of image coords
                vs.uv_star = vs.pf;
            else
                if ~isempty(vs.Tf)
                    vs.Tf = transl(0, 0, 1);
                    warning('setting Tf to default');
```



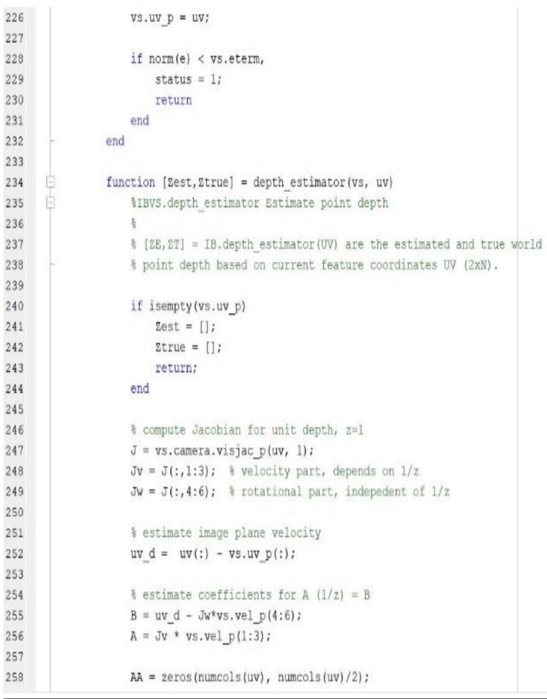
```
            end
            % final pose is specified in terms of a camera-target pose
            %   convert to image coords
            vs.uv_star = vs.camera.project(vs.P, 'Tcam', inv(vs.Tf));
        end

        % initialize the vservo variables
        vs.camera.T = vs.T0;    % set camera back to its initial pose
        vs.Tcam = vs.T0;                % initial camera/robot pose

        % show the reference location, this is the view we wish to achieve
        % when Tc = Tct_star


        vs.vel_p = [];
        vs.uv_p = [];
        vs.history = [];
    end

    function status = step(vs)
        %IBVS.step Simulate one time step
        %
        % STAT = IB.step() performs one simulation time step of IBVS.  It is
        % called implicitly from the superclass run method.  STAT is
        % one if the termination condition is met, else zero.
        %
        % See also VisualServo, IBVS.run.

        status = 0;
        Zest = [];

        % compute the view
```



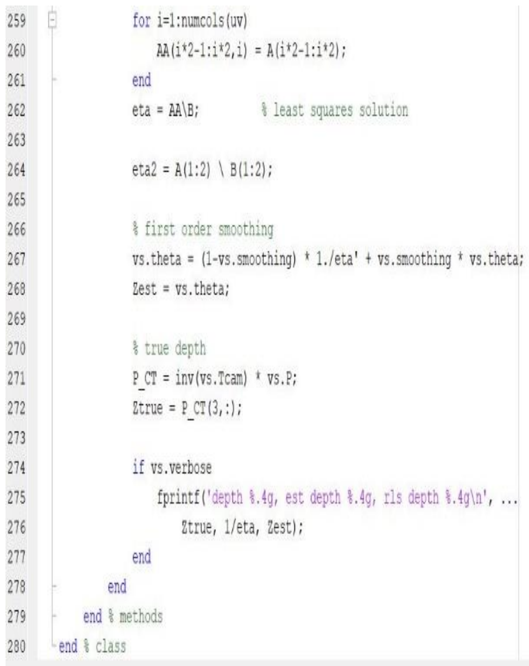
```
        uv = vs.camera.plot(vs.P);

        % optionally estimate depth
        if vs.depthest
            % run the depth estimator
            [Zest,Ztrue] = vs.depth_estimator(uv);
            if vs.verbose
                fprintf('Z: est=%f, true=%f\n', Zest, Ztrue)
            end
            vs.depth = Zest;
            hist.Ztrue = Ztrue(:);
            hist.Zest = Zest(:);
        end

        % compute image plane error as a column
        e = uv - vs.uv_star;   % feature error
        e = e(:);


        % compute the Jacobian
        if isempty(vs.depth)
            % exact depth from simulation (not possible in practice)
            pt = inv(vs.Tcam) * vs.P;
            J = vs.camera.visjac_p(uv, pt(3,:) );
        elseif ~isempty(Zest)
            J = vs.camera.visjac_p(uv, Zest);
        else
            J = vs.camera.visjac_p(uv, vs.depth );
        end

        % compute the velocity of camera in camera frame
        try
```



```
            v = -vs.lambda * pinv(J) * e;
        catch
            status = -1;
            return
        end

        if vs.verbose
            fprintf('v: %.3f %.3f %.3f %.3f %.3f %.3f\n', v);
        end

        % update the camera pose
    Td = SE3(trnorm(delta2tr(v)));    % differential motion
        %Td = expm( skewa(v) );
        %Td = SE3( delta2tr(v) );
        vs.Tcam = vs.Tcam .* Td;       % apply it to current pose
    %vs.Tcam = trnorm(vs.Tcam);

        % update the camera pose
        vs.camera.T = vs.Tcam;

        % update the history variables
        hist.f = uv(:);
        vel = tr2delta(Td);
        hist.vel = vel;
        hist.e = e;
        hist.en = norm(e);
        hist.jcond = cond(J);
        hist.Tcam = vs.Tcam;

        vs.history = [vs.history hist];

        vs.vel_p = vel;
```



```
        vs.uv_p = uv;

        if norm(e) < vs.eterm,
            status = 1;
            return
        end
    end

    function [Zest,Ztrue] = depth_estimator(vs, uv)
        %IBVS.depth_estimator Estimate point depth
        %
        % [ZE,ZT] = IB.depth_estimator(UV) are the estimated and true world
        % point depth based on current feature coordinates UV (2xN).

        if isempty(vs.uv_p)
            Zest = [];
            Ztrue = [];
            return;
        end

        % compute Jacobian for unit depth, z=1
        J = vs.camera.visjac_p(uv, 1);
        Jv = J(:,1:3);  % velocity part, depends on 1/z
        Jw = J(:,4:6);  % rotational part, indepedent of 1/z

        % estimate image plane velocity
        uv_d =  uv(:) - vs.uv_p(:);

        % estimate coefficients for A (1/z) = B
        B = uv_d - Jw*vs.vel_p(4:6);
        A = Jv * vs.vel_p(1:3);

        AA = zeros(numcols(uv), numcols(uv)/2);
```



```
            for i=1:numcols(uv)
                AA(i*2-1:i*2,i) = A(i*2-1:i*2);
            end
            eta = AA\B;         % least squares solution

            eta2 = A(1:2) \ B(1:2);

            % first order smoothing
            vs.theta = (1-vs.smoothing) * 1./eta' + vs.smoothing * vs.theta;
            Zest = vs.theta;

            % true depth
            P_CT = inv(vs.Tcam) * vs.P;
            Ztrue = P_CT(3,:);

            if vs.verbose
                fprintf('depth %.4g, est depth %.4g, rls depth %.4g\n', ...
                    Ztrue, 1/eta, Zest);
            end
        end
    end % methods
end % class
```

REFERENCES

[1] E. R. Davies, “Automated Visual Inspection,” *Mach. Vis.*, p. 626, 2005, doi: 10.1016/b978-0-12-206093-9.50067-8.

[2] Z. M. Bia and L. Wang, “Advances in 3D data acquisition and processing for industrial applications,” *Robotics and Computer-Integrated Manufacturing*, vol. 26, no. 5. Elsevier Ltd, pp. 403–413, Oct. 01, 2010, doi: 10.1016/j.rcim.2010.03.003.

[3] E. N. Malamas, E. G. M. Petrakis, M. Zervakis, L. Petit, and J. D. Legat, “A survey on industrial vision systems, applications and tools,” *Image and Vision Computing*, vol. 21, no. 2. Elsevier Ltd, pp. 171–188, Feb. 10, 2003, doi: 10.1016/S0262-8856(02)00152-X.

[4] T. Brosnan and D. W. Sun, “Improving quality inspection of food products by computer vision - A review,” *J. Food Eng.*, vol. 61, no. 1 SPEC., pp. 3–16, 2004, doi: 10.1016/S0260-8774(03)00183-3.

[5] T. S. Newman, “Survey of automated visual inspection,” *Comput. Vis. Image Underst.*, vol. 61, no. 2, pp. 231–262, 1995, doi: 10.1006/cviu.1995.1017.

[6] A. De Luca, G. Oriolo, and P. Robuffo Giordano, “Feature Depth Observation for Image-based Visual Servoing: Theory and Experiments,” *Int. J. Rob. Res.*, vol. 27, no. 10, pp. 1093–1116, Oct. 2008, doi: 10.1177/0278364908096706.

[7] E. Malis and P. Rives, “Robustness of image-based visual servoing with respect to depth distribution errors,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2003, vol. 1, pp. 1056–1061, doi:

10.1109/robot.2003.1241732.

[8] W. Bachta and A. Krupa, “Towards ultrasound image-based visual servoing,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2006, vol. 2006, pp. 4112–4117, doi: 10.1109/ROBOT.2006.1642334.

[9] G. Morgenthal, N. Hallermann, and U. Kingdom, “(UAV) Based Visual Inspection of Structures,” *Adv. Struct. Eng.*, vol. 17, no. 3, 2014.

[10] J. Doyle, “Robust and optimal control,” *Proceedings of the IEEE Conference on Decision and Control*, vol. 2. pp. 1595–1598, 1996, doi: 10.1016/s0005-1098(97)00132-5.

[11] L. D. Berkovitz, “Optimal Control Theory,” *Am. Math. Mon.*, vol. 83, no. 4, p. 225, 1976, doi: 10.2307/2318209.

[12] D. Blackwell, “Discrete Dynamic Programming,” 1962.

[13] G. L. Foresti, “Visual inspection of sea bottom structures by an autonomous underwater vehicle,” *IEEE Trans. Syst. Man, Cybern. Part B Cybern.*, vol. 31, no. 5, pp. 691–705, Oct. 2001, doi: 10.1109/3477.956031.

[14] B. Silveira, R. Melo, and D. B. Costa, “Using UAS for Roofs Structure Inspections at Post-occupational Residential Buildings,” in *Lecture Notes in Civil Engineering*, vol. 98, Springer, 2021, pp. 1055–1068.

[15] Y. Liu, J. K. W. Yeoh, and D. K. H. Chua, “Deep Learning-Based Enhancement of Motion Blurred UAV Concrete Crack Images,” *J. Comput. Civ. Eng.*, vol. 34, no. 5, Sep. 2020, doi: 10.1061/(ASCE)CP.1943-5487.0000907.

[16] V. T. Hoang, M. D. Phung, T. H. Dinh, and Q. P. Ha, “System Architecture for

Real-Time Surface Inspection Using Multiple UAVs," *IEEE Syst. J.*, vol. 14, no. 2, pp. 2925–2936, Jun. 2020, doi: 10.1109/JSYST.2019.2922290.

[17] S. Huang and Y. Pan, "Computers in Industry Automated visual inspection in the semiconductor industry : A survey," *Comput. Ind.*, vol. 66, pp. 1–10, 2015, doi: 10.1016/j.compind.2014.10.006.

[18] A. Mital, M. Govindaraju, and B. Subramani, "A comparison between manual and hybrid methods in parts inspection," *Integr. Manuf. Syst.*, vol. 9, no. 6, pp. 344–349, 1998, doi: 10.1108/09576069810238709.

[19] Y. Ham, K. K. Han, J. J. Lin, and M. Golparvar-Fard, "Visual monitoring of civil infrastructure systems via camera-equipped Unmanned Aerial Vehicles (UAVs): a review of related works," *Vis. Eng.*, vol. 4, no. 1, pp. 1–8, 2016, doi: 10.1186/s40327-015-0029-z.

[20] Z. Xiao-bo, Z. Jie-wen, L. Yanxiao, and M. Holmes, "In-line detection of apple defects using three color cameras system," *Comput. Electron. Agric.*, vol. 70, no. 1, pp. 129–134, 2010, doi: 10.1016/j.compag.2009.09.014.

[21] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 3951 LNCS, pp. 430–443, 2006, doi: 10.1007/11744023_34.

[22] A. Ferreira, C. Cassier, and S. Hirai, "Automatic microassembly system assisted by vision servoing and virtual reality," *IEEE/ASME Trans. Mechatronics*, vol. 9, no. 2, pp. 321–333, 2004, doi: 10.1109/TMECH.2004.828655.

[23] J. Salvi, X. Armangué, and J. Batlle, "A comparative review of camera calibrating

methods with accuracy evaluation,” *Pattern Recognit.*, vol. 35, no. 7, pp. 1617–1635, Jul. 2002, doi: 10.1016/S0031-3203(01)00126-1.

[24] G. L. Mariottini, G. Oriolo, and D. Prattichizzo, “Image-based visual servoing for nonholonomic mobile robots using epipolar geometry,” *IEEE Trans. Robot.*, vol. 23, no. 1, pp. 87–100, Feb. 2007, doi: 10.1109/TRO.2006.886842.

[25] D. Lee, T. Ryan, and H. J. Kim, “Autonomous landing of a VTOL UAV on a moving platform using image-based visual servoing,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2012, pp. 971–976, doi: 10.1109/ICRA.2012.6224828.

[26] D. Lee, H. Lim, H. J. Kim, Y. Kim, and K. J. Seong, “Adaptive image-based visual servoing for an underactuated quadrotor system,” *J. Guid. Control. Dyn.*, vol. 35, no. 4, pp. 1335–1353, 2012, doi: 10.2514/1.52169.

[27] G. Allibert, E. Courtial, and F. Chaumette, “Predictive control for constrained image-based visual servoing,” *IEEE Trans. Robot.*, vol. 26, no. 5, pp. 933–939, 2010, doi: 10.1109/TRO.2010.2056590.

[28] C. Gao, X. Piao, and W. Tong, “Optimal motion control for IBVS of robot,” *Proc. World Congr. Intell. Control Autom.*, pp. 4608–4611, 2012, doi: 10.1109/WCICA.2012.6359352.

[29] C. To and H. A. L. Id, “Potential problems of stability and convergence in image-based and position-based visual servoing,” pp. 66–78, 2009, doi: 10.1007/BFb0109663.

[30] L. Deng, F. Janabi-Sharifi, and W. J. Wilson, “Stability and robustness of visual

servoing methods," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2, pp. 1604–1609, 2002, doi: 10.1109/ROBOT.2002.1014772.

[31] E. Malis, "Improving vision-based control using efficient second-order minimization techniques," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2004, vol. 2004, no. 2, pp. 1843–1848, doi: 10.1109/robot.2004.1308092.

[32] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, 1968, doi: 10.1109/TSSC.1968.300136.

[33] M. Pollack and W. Wiebenson, "Solutions of the Shortest-Route Problem—A Review," *Oper. Res.*, vol. 8, no. 2, pp. 224–230, Apr. 1960, doi: 10.1287/opre.8.2.224.

[34] Q. Zhu, Y. Liu, J. Lu, and J. Cao, "On the optimal control of boolean control networks," *SIAM J. Control Optim.*, vol. 56, no. 2, pp. 1321–1341, Apr. 2018, doi: 10.1137/16M1070281.

[35] J. L. Pedersen and G. Peskir, "Constrained dynamic optimality and binomial terminal wealth," *SIAM J. Control Optim.*, vol. 56, no. 2, pp. 1342–1357, 2018, doi: 10.1137/16M1085097.

[36] X. Yang, B. Chen, Y. Li, Y. Liu, and F. E. Alsaadi, "Stabilization of dynamic-algebraic Boolean control networks via state feedback control," *J. Franklin Inst.*, vol. 355, no. 13, pp. 5520–5533, Sep. 2018, doi: 10.1016/j.jfranklin.2018.05.049.

[37] Y. Wu, X. M. Sun, X. Zhao, and T. Shen, "Optimal control of Boolean control

networks with average cost: A policy iteration approach," *Automatica*, vol. 100, pp. 378–387, Feb. 2019, doi: 10.1016/j.automatica.2018.11.036.

[38] D. Geiger, A. Gupta, J. Vlontzos, and L. A. Costa, "Dynamic Programming for Detecting, Tracking, and Matching Deformable Contours," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, no. 3, pp. 294–302, 1995, doi: 10.1109/34.368194.

[39] A. Drouot, R. Zhao, L. Irving, D. Sanderson, and S. Ratchev, "Measurement Assisted Assembly for High Accuracy Aerospace Manufacturing," *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 393–398, Jan. 2018, doi: 10.1016/j.ifacol.2018.08.326.

[40] A. Angerer, C. Ehinger, A. Hoffmann, W. Reif, and G. Reinhart, "Design of an automation system for preforming processes in aerospace industries," in *IEEE International Conference on Automation Science and Engineering*, 2011, pp. 557–562, doi: 10.1109/CASE.2011.6042411.

[41] V. Serebrenny, D. Lapin, A. Mokaeva, and M. Shereuzhev, "Technological collaborative robotic systems," in *AIP Conference Proceedings*, Nov. 2019, vol. 2171, no. 1, p. 170008, doi: 10.1063/1.5133319.

[42] J. Ashour, "Why Do Aircraft Have Wiring Harnesses? - InterConnect Wiring." https://www.interconnect-wiring.com/blog/why-do-aircraft-have-wiring-harnesses/ (accessed May 02, 2020).

[43] R. E. Larson, "Dynamic Programming with Reduced Computational Requirements," *IEEE Trans. Automat. Contr.*, vol. 10, no. 2, pp. 135–143, 1965, doi: 10.1109/TAC.1965.1098129.

[44] Y. Zhou, E. J. Van Kampen, and Q. P. Chu, “Incremental approximate dynamic programming for nonlinear adaptive tracking control with partial observability,” *J. Guid. Control. Dyn.*, vol. 41, no. 12, pp. 2554–2567, 2018, doi: 10.2514/1.G003472.

[45] Y. Zhou, E.-J. Van Kampen, and Q. P. Chu, “An Incremental Approximate Dynamic Programming Flight Controller Based on Output Feedback,” Jan. 2016, doi: 10.2514/6.2016-0360.