CNC Alarm Resolution and Work Cell Monitoring with a Robotic Arm

A Major Qualifying Project Report submitted to the faculty of

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the degree of Bachelor of Science

Submitted By:

_____

Brian Francis

Robotics Engineering

_____

Mayank Govilla

Robotics Engineering and Computer Science

_____

Nikolas Neathery

Robotics Engineering

May 3, 2023

_____

Prof. Siavash Farzan, Advisor

Prof. Greg Lewin, Advisor

Prof. Jing Xiao, Advisor

Robotics Engineering Department

Abstract

Computer Numeric Control (CNC) revolutionized the manufacturing industry by enabling the creation of highly accurate and complex parts at high speeds. However, CNC systems have been heavily reliant on human operators to oversee and control the manufacturing process since their creation. Despite recent advancements in automating CNC milling, existing approaches have overlooked the need for a methodology to handle errors and failures in the system.

To address this gap, we developed a set of Robot Operating System (ROS) nodes that monitor and autonomously manage a CNC while reacting to alarms in real-time with the hope of reducing downtime during lights our manufacturing. At any point during normal operation a CNC could be stopped by an alarm halting production. Our system is designed to reduce this downtime by resolving the present alarms. Our approach additionally tracks a variety of performance metrics, allowing for later evaluation and continuous improvement of the system via the Flexxbotics environment. The methodology we created is scalable and adaptable to a wide range of CNC systems and can be customized and configured to fit the specific needs of different manufacturers. Because the framework is adaptable to other CNC systems, our methodology is a valuable tool for manufacturers looking to improve their efficiency and productivity.

Acknowledgements

We would like to thank our sponsors, Flexxbotics, for providing us with the technology to be able to actualize this work. We would also like to thank our sponsor contacts Zac Chupka and Megan Hiatt for their technical assistance and wisdom throughout the course of this project, as well as our faculty advisors Professors Siavash Farzan, Greg Lewin, and Jing Xiao.

Authorship

Brian Francis created the motion application as well as the alarm detection and resolution systems. Mayank Govilla integrated the recording information into the web interface. Nikolas Neathery set up the monitoring system and runtime analysis. All authors drafted, read, and approved the report.

Table of Contents

List of Figures

List of Tables

Executive Summary

The use of Computer Numeric Control (CNC) in manufacturing processes is heavily reliant on human operators, making it highly vulnerable to issues that stem from human fallibility. Consequently, the occurrence of issues on a production line significantly reduces efficiency and increases the likelihood of production injuries, resulting in adverse impacts on both people and the economy. Given the high cost of production loss and injury, enhancing CNC workflows is a critical priority for organizations. Two effective strategies for minimizing workplace accidents and increasing output involve analyzing performance metrics to identify recurrent failures that impede production and automating previously inefficient procedures. We sought to find a way to implement this at the level of a work cell.

Our aim was to devise a system to monitor the workflow of a work cell, which encompasses machines, equipment, and workers involved in a specific production step, (in our case a CNC machine and robotic arm), and simultaneously automate the tasks that were previously assigned to human operators. We designed the system to be integrated across as many brands and manufacturers as possible to meet the needs of the most possible users. To make the system more general, we implemented our solution using the Robot Operating System (ROS), an open-source platform for robotic design and testing that can be used by nearly every major robotic arm manufacturer.

To develop, test, and demonstrate our system, we used a Universal Robotics (UR) 5e arm and a HAAS Super Mini Mill as our production devices. We controlled the arm with ROS commands via an external controller and operated the CNC mill in its standard configuration without significant alterations. Our experiment entailed the robot arm inserting and removing a blank metal piece into and from the CNC machine in a loop, simulating a typical CNC job. During this task, we demonstrated the system's capability to monitor the process in real-time, report the data to the Flexxbotics environment, and resolve a subset of errors from the CNC machine.

To record and monitor performance analytics in a system, we used the FlexxEdge, which is a peripheral device created by FlexxBotics. The FlexxEdge networked together with the CNC, the robot arm, and the overall job controller. Over this network we passed any relevant performance metrics or runtime information that could be used to make the system more reactive including cycle times, how many parts have been created, the current tasks of the system, and digital signals into and out of each piece of hardware. This information was collected and centralized into the FlexxConnect application that serves as a hub for data aggregation. By successfully integrating our test workflow into the FlexxBotics environment, we demonstrated a workflow that is largely brand agnostic and can be used to monitor previously untracked brands' and manufacturers' performance.

The FlexxEdge also enabled the error handling service we created to react to a selection of errors from the CNC machine. To trigger error handling, the FlexxEdge sent CNC status info to the robot controller, including the emergency light output, which indicated an alarmed state.

Then the UR5e would resolve errors, if possible, based on preset categories like improper door position, tool head, program load failure, or a basic system reset.

Through our testing we found that we were able to develop a functional machine tending application that can resolve 14 alarm codes autonomously. In addition, the development of a system agnostic runtime monitoring system allows for ROS based devices to be integrated into the FlexxBotics environment.

Chapter 1: Introduction

Every year sees a growing demand for more and more goods, and manufacturing plants are being required to produce at an ever-increasing rate. Since every wasted second means wasted cost, companies are constantly developing and refining their processes to be optimally productive. Their goal is to have a system that can complete cycles as quickly and consistently as possible, while maintaining their standard of quality assurance and precision.

One of the primary tools the industry can use to improve a manufacturing process is automation. Robots can move faster and more precisely than humans, and introducing automation sees cost reduction in the majority of cases (*Save-to-Transform as a Catalyst for Embracing Digital Disruption | Deloitte China | Strategy & Operations*, n.d.). A robot is also able to do things a human cannot do such as working in dangerous environments or lifting heavy payloads. Therefore, manufacturers place a high value on being able to automate as many steps of a process as possible. The ability to automate more tasks in a process enables manufacturers to complete a larger number of cycles without downtime from waiting on human intervention. However, one of the major challenges of automating any task is matching the perceptiveness and awareness of a human operator. Robots struggle to perceive and recognize objects in varying lighting conditions, especially when the objects are reflective or transparent. Another limitation is the ability to interpret human actions and intentions, making it challenging for robots to work safely and effectively alongside humans in collaborative environments. (*Michael*, 2020) The way roboticists and manufacturers have been attempting to resolve this problem is by making factories "smarter" by adding greater computing and communication power to production lines.

The desire to have more connected devices led to the creation of the Internet of Things (IoT). The IoT is the name given to how devices like production lines, computers, sensors, and other hardware have become interconnected via advances in embedded processing and software. Sensors and data processing pipelines provide visibility into a facility's operations for improved fault detection, safety, and overall efficiency (*Syafrudin et al.*, 2018). This interconnectivity allows for the systems of an operation to be monitored remotely from a central location. By having the control center for multiple machines in one place, a supervisor can oversee many processes at the same time and in greater detail. Each of these processes are broken down into work cells. A work cell is defined as the arrangement of resources and machines, most often a combination of people, equipment, and materials, designed to improve the quality and speed of a particular output (*admin*, 2016). Generally, a work cell is one part of a larger process and refers to all the necessary elements to complete one or more steps in a production process. A work cell could be a car body and the painters who spray it, or it could be a conveyor and hydraulic press that stamps sheet metal.

The work cell that this project will focus on is a Computer Numerical Control (CNC) machine tending work cell. Industrial CNC production is characterized by its ability to produce highly accurate and complex parts, making it a popular choice for mass production when precision is required. Compared to other forms of manufacturing, CNC production also offers greater consistency, as well as increased efficiency and reduced waste. However, the process produces parts in a much lower volume than a less complex form of production like stamping or

casting. A typical CNC tending work cell will include a CNC machining mill with an operator who has access to the control panel as well as the raw materials that will become the part. The operator will place material inside of the mill where the CNC will cut the desired final product out of the initial material. Then the completed part will be removed again by the human operator and taken to the next step of production.

Unlike other manufacturing processes, CNC machine tending does not have many commercial options for automation. Typically, there is a team of operators monitoring and managing the CNC machines in addition to the administrative and management teams. The presence of these operators in work cells comes at a large labor cost as well as creating the risk for injury from production accidents. We believe that a more intelligent CNC work cell would create opportunities for automation to reduce the required operational manpower for greater overall optimization.

Work cell management is the sector of the manufacturing industry that seeks to use the IoT to analyze and further optimize workflows. As mentioned earlier, companies will use a centralized information hub to operate production remotely and monitor the status of every machine. Because of this recording hub, manufacturers can organize their factories without needing someone to directly watch every single work cell. The most common existing management offerings are primarily a high-level procedure monitoring service. Companies like Flexxbotics, Tulip, and SCYTEC provide cloud services that can track a variety of runtime information about a work cell and its cycles without providing any ability to control or intervene. This style of monitoring is typically coupled with some form of additional analysis software that can help a company to identify recurrent system failures or particularly troubling patterns of downtime. While straightforward, the value provided by these services comes from their convenience. Instead of requiring someone to directly notice when errors and downtime occur, all the information is sent directly to the cloud in a centralized hub automatically where management can decide how to react. This means less maintenance and monitoring staff are required, but it still requires staff on duty to resolve any errors that arise.

The same technology that has enabled work cell management is also responsible for many of the advancements in industrial automation. For example, the Amazon warehouses use package carrying robots and smart shelving to manage their large and diverse inventories. They have a system that knows where each package is and what is in it because they have a network of interconnected devices in each warehouse. That level of automation has allowed for the process of packaging and delivery to be reduced to days instead of weeks. As it pertains to our focus of CNC machine tending, there are existing automated work cells, but they are typically limited to material loading or cycle control. These autonomous cells also still require significant assistance from a human operator due to the variety and complexity of tasks required to interface with a CNC machine. Therefore, it is more of an augmentation rather than a replacement of the operator. The closest available option for autonomous CNC tending is a solution from Robotiq, who has presented a work cell management system that can move material, start and stop cycles, and control the workholding by using a variety of actuators attached to the existing control surfaces and a robotic arm. (*Machine Tending Solution*, n.d.) This allows for a cell to be run almost completely without human intervention in good

conditions. However, this is still limited in terms of error resolution in the case that an alarm occurs.

The Robotiq solution has been successful enough to see commercial use, but researchers have been looking into ways to integrate the functions of the individual component actuators into the robotic arm that is more flexible than basic buttons or switches. Having an arm integrated so intimately with a complex system like a CNC has its own issues though. Currently, the leading state of the art solution is to point a camera at the human–machine interface (HMI) display to detect the state of the machine. Since the state information displayed on these tablets is complex researchers have used deep learning to develop neural networks for recognizing the text on the screen (*Jia et al.*, 2022, *Jia et al.*, 2021). These algorithms achieved high levels of state recognition but must be retrained for different HMIs and are difficult to update or edit for an end user. At the current state these solutions are not generalizable for the wide variety of machine brands that are used in production. Additionally, the trained models are not user friendly to update in the event of a software update or other visual system change. In order to use a deep learning algorithm companies would need to teach employees how to retrain and update the software's neural networks, which would likely be both time consuming and costly due to the complexity. While the CNC has over 1000 unique alarm codes (HelmanCNC, 2013) many of them are unresolvable by a robotic operator. For this project many of the most difficult alarm codes were excluded from the scope in order to prioritize the wider range of more feasible tasks. This excluded alarms that might require screwdriving or jogging the machine which is technically possible using a robotic arm but would require a high degree of precision and tuning.

Figure 1.1: Example work cell with a robot tending to a CNC

Our goal is to create a seamless work cell operation that capitalizes on the advancements in collaborative robotics and the growing interconnectivity of the industrial workspace, without adding the complication of numerous peripherals or deep visual

processing. With only a robotic arm and an intelligent, flexible control platform we will be able to create a highly efficient cell, like that seen Fig 1.1, that avoids the drawbacks of having a human operator. To meet our design goals, we are developing for the Flexxbotics platform that can aggregate data from both the CNC and robotic arm in a work cell. Rather than just taking parts in and out of a machine, we will provide the ability for the robot to respond to messages and alarm codes thrown by the CNC, while also reporting run record statistics about the current job to the cloud. The system that is developed will be able to resolve alarms presented by the CNC autonomously, handle the basic operational tasks of running a CNC, and provide important system information to a system wide cloud service. Not only will this be able to improve production efficiency by decreasing downtime, but it will also provide important insight into the overall system that can be used to improve total efficiency.

Chapter 2: Background

According to the International Labor Organization (ILO) there are more than 2 million work related fatalities a year (Caribbean, 2004) in addition to the 270 million annual occupational injuries (Hämäläinen et al., 2006). This is not only a massive and unnecessary endangerment of human life, but also comes as a major fiscal cost with close to $5 billion being lost every year (Mitchell & University, 1996) from accidents and injuries. Even one case of illness takes on a cost of $17,000 to individuals, employers, and society (*Statistics - Costs to Britain of Workplace Injuries and New Cases of Work-Related Ill Health*, n.d.) This makes health and safety an absolute priority when considering ways to improve system efficiency. If you can reduce the number of injuries that can occur, then you will not only be saving money but improving overall health and quality of life for your employees.

Autonomy offers an effective technological solution for companies to improve workplace safety and improve productivity. Since machines can perform highly repetitive tasks with a greater precision than humans and record their performance for future studies, an autonomous system can reduce waste, improve consistency, and provide access to important data and performance metrics (*deSpautz*, 1994). Because of this autonomy is a staple of nearly every industry with 76% of manufacturers using autonomy as a central element of their production floor. (Duong et al., 2020), (Katana, 2019) Autonomy itself takes many forms, with the focus of this work being material handling. In the context of manufacturing, material handling refers to the physical motion of parts and materials throughout a system. This could be as simple as a conveyor belt that carries parts between machines or as advanced as a robotic arm that manipulates a work fixture while it is painted (Annem et al., 2019). Autonomous material handling is critical to ensure high speed throughput of parts and eliminating the otherwise impossible or dangerous tasks that can so commonly lead to injury. Additionally, since material handling is also a non-productive step, any time spent on moving parts between production elements is a detriment to overall system speed and cost.

CNC machines are largely self-contained and often can be the only step a part goes through. The only movement required to operate a CNC is the loading and unloading of parts to the mill which is referred to as machine tending. The most common way this is done is by having a human operator who supervises the CNC cycle and handles the workpiece. They are responsible for placing fresh blanks and then subsequently removing completed parts in the machine. Autonomous machine tending can improve cycle times, save space on the production floor, and allow for auxiliary processes such as quality assurance to be handled inside your work cell. (*The Advantages of Automated Machine Tending - Fanuc*, n.d.) Autonomous machine tending is a well-developed field that recently has seen a growth of interest. (Annem et al., 2019) The most advanced academic approaches include mobile robots that can tend multiple cells at the same time (Al-Hussaini et al., 2020), and systems that learn and improve their own performance. (Jia et al., 2020)

While there is a range of applications that handle the primary function of machine tending, a major limitation of the existing methods is the lack of ability react to downtime in production. Either for safety reasons or technological limitations, machine tending applications rarely handle their own errors. The ability to manage faults is typically delegated to a human

supervisor or operator. (*The Cognitive Costs and Benefits of Automation*, n.d.) Even if the supervisor is immediately aware of any downtime there is still a period where the CNC is unable to produce while it waits for help to resume operation. However, if a machine tending application were able to resolve its own faults, the cumulative downtime would decrease, and costly losses of productivity would be much less likely. As such, there has been recent work in developing automated systems that can react to their own errors instead of requiring regular human assistance. One possible way to do this is to add a camera into the system that watches the Human Machine Interface (HMI) and then discerns status information. (Jia et al., 2021), (Jia et al., 2022) The visual data from the camera is passed into a deep learning-based text recognition method that identifies the working status of the CNC. Depending on the state of the CNC the robot can complete its own intervention without ever needing to get a human operator involved.

Instead of having to develop complex networks to turn visual information into system information, we propose using a peripheral device that connects the CNC and robot controllers. These kinds of devices have been created to track production analytics by several companies including our sponsor. (*CNC Machine Monitoring & OEE Software*, n.d.), (*FlexxCNC - UR to CNC Communication Interface*, n.d.) A work cell monitor can provide much of the same information but instead of pulling data from a control terminal or screen like a human, it gets access to it from the system itself. Due to the level of integration required to make this function these applications are typically developed on a system-by-system basis, and they are very difficult to generalize. However, the ROS Industrial interface provides an opportunity to create a system agnostic application for work cell management and alarm intervention.

The purpose of using ROS for this project is to create a generic data aggregation platform that could work on many different robots without the need for custom integrations. The Robot Operating System (ROS) is a set of open-source packages that are meant to unify robot development, especially on multiple devices. It has become very popular with robot developers with the goal of having a more standardized way to create robotic applications. The ROS architecture is based on Nodes that are registered to a ROS master that communicate using a publisher/subscriber model on channels called topics. A Node can subscribe or publish to any number of topics that have a specific data type called a message as seen in the example in Fig 2.1. (*ROS: Home*, n.d.)

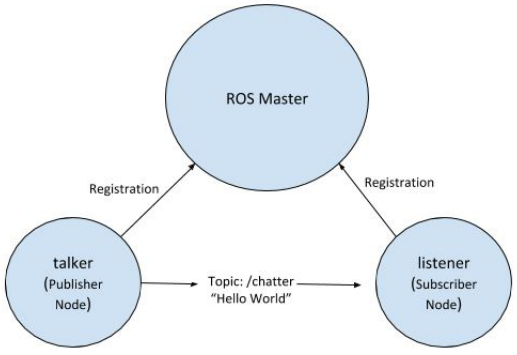

Figure 2.1: ROS Topic communication from an example package

ROS Industrial is a coalition of several robotics companies and manufacturers that aimed to create a generic interface to develop manufacturing applications. These vendors include Universal Robots (UR), ABB, FANUC, and many others. Not only is ROS widely used, but due to ROS Industrial there is a series of standards and conventions that make integration between different brands mostly seamless. (*ROS-Industrial*, 2023) Fig 2.2 describes the connections and functionality of the interface including file types, communication styles, and shared packages.

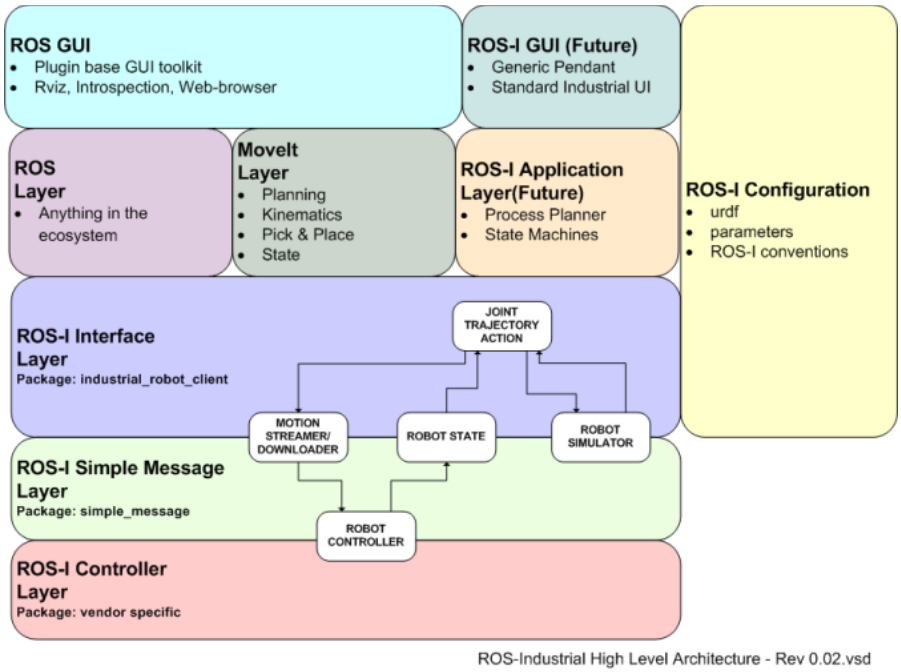

Figure 2.2: Layer Architecture of ROS Industrial

At the bottom of this diagram are the ROS-I Controller, Message, and Interface layers. These layers are how the industrial robot packages can retain a common interface for receiving data about robots and sending control commands. For example, the UR_Robot_Driver (the UR implementation of the ROS Industrial Interface and the interface this project uses) implements these three layers and exposes the generic robot state to ROS as a topic. Then, a set of nodes can be built on top of the drivers to parse the information and aggregate the necessary data.

By building our monitoring and error resolution platform on top of the ROS Industrial Interface, all the robot manufacturers that implement the interface could slot into the system with minimal effort, serving the goal of creating a generic package.

Chapter 3: Design and Development

To be an effective autonomous solution for CNC machine tending, our developed robotic system needs to be able to perform the following tasks:

1. The arm should be able to perform material loading, part unloading, and error handling during the CNC job.
2. The cell should be able to report and record runtime information using the ROS messaging system so a single, centralized cloud application can monitor the status of the work cell.
3. Demonstrate the working system with the Universal Robotics UR5e arm and the HAAS Super Mini Mill.

This design was built upon the FlexxEdge, a device Flexxbotics developed to publish machine states to the cloud. To measure the success of our solution we evaluated performance on flexibility, relative performance to humans, safety, and consistency.

3.1 System Overview

Fig 3.1 illustrates the high-level flow of information throughout the work cell. Both the CNC and the arm connected via the Flexxedge, so that we can interpret and react to signals from either device. The CNC is connected to the network via Wi-Fi, and the UR5e is wired directly to the FlexxEdge via ethernet. The CNC and robot are unable to send messages to each other directly, but with the FlexxEdge serving as a communication bus in the middle it becomes possible for both side of the system to share information to a job controller. Access to the overarching work cell management service called FlexxControl, is also facilitated through the FlexxEdge via the onboard webapp FlexxConnect. The basic flow of information is that a job will be loaded onto the work cell through the FlexxEdge that describes the normal tending operations like the loading/unloading positions. Whenever the Haas CNC throws an alarm, the alarm handler service will format state information from the CNC into instructional information for the UR5e, and then the UR5e will receive the instructions and complete the error resolution. During this process, the FlexxEdge will also be recording the downtime and system diagnostics and reporting to the FlexxControl via the FlexConnect application.

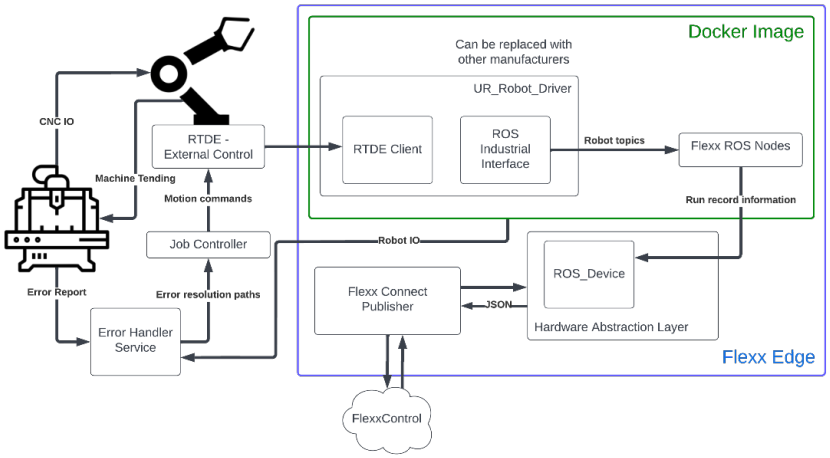

Figure 3.1: Architecture diagram of the work cell

3.1.1 FlexxEdge

The primary purpose of the FlexxEdge is to record and report runtime information about a work cell. It does this by acting as the communication bridge between the production devices and the cloud. It parses and formats the data stream from its connections and then publishes in real time to the FlexxConnect. The information it reports is stored in the form of a *run record* which is a packet of relevant cycle information such as status, failure count, part count, and cycle time. The FlexxEdge currently only supports a URScript to access information about specific Universal Robot arms. Our goal is to enable the FlexxEdge to create and publish information through ROS so that the device would be able to integrate with a wider range of existing system architectures more seamlessly. As discussed in Chapter 2, several industrial robot manufacturers have already adopted the ROS platform, so being able to interface with it will extend the FlexxEdge to any robot that implements the ROS Industrial Interface.

The FlexxEdge had three elements of development for this project. First, we created a set of data aggregation and reporting nodes in ROS to run on the Edge. Their purpose is to parse and package information from ROS Industrial for reporting purposes. Second, we had to configure the hardware abstraction layer (HAL) to be able to receive information from ROS. Once the HAL was able to interpret information coming from ROS, then we were able to pass packets to the FlexxControl environment for analysis by integrating into their web application.

3.1.2 Universal Robotics UR5e Arm

The next major component of the work cell is the tending robot. We chose to use the UR5e arm for a few major reasons. First, the UR5e is a cobot which means it is a robotic arm meant to work alongside humans. It has built-in safety stops, low load capacity, and one of the most mature ROS libraries. The UR5e is also large enough to reach both the inside of the Haas Mini Mill's workspace and control panel. In addition, our end of arm tooling had to be capable of interfacing with buttons, doors, material, and tools.

The next major reason why UR5e worked well for this project was for its mature ROS support with the UR_ROS_Driver and MoveIt libraries. MoveIt is an open-source motion planning framework for ROS that is commonly used with ROS Industrial robots because it is highly configurable and provides the ability to perform motion planning with collision avoidance. The entire work cell can be configured in MoveIt in a simulated manner, and since the environment is quite congested, it was important to be able to navigate safely to prevent damage. So, the development pipeline was based around using the MoveIt library to plan waypoints that move the arm through the environment, which were then recorded and published to the arm whenever that resolution was the appropriate reaction.

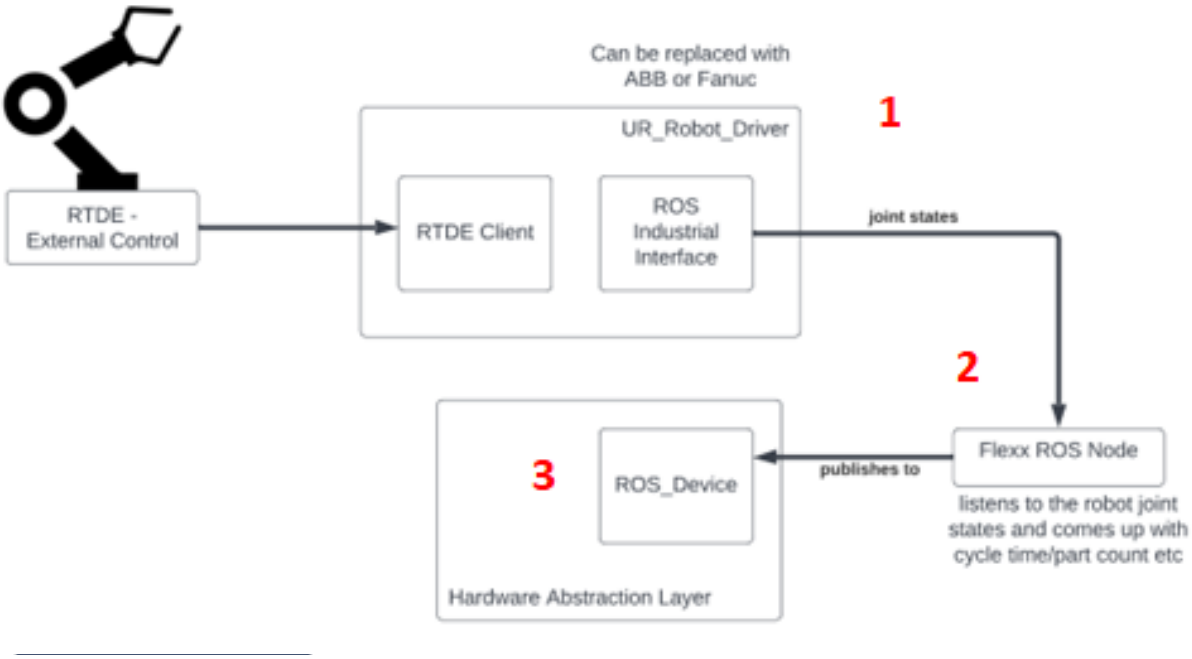

Figure 3.2: ROS architecture for the UR Driver

The UR robot driver is simply a translation between UR's proprietary Real Time Data Exchange (RTDE) protocol and the ROS Industrial Interface (see Fig 3.2, 1). We can then run ROS nodes on our work cell monitor that consume the data denoted by the Interface to construct an archive of system status information. (Fig 3.2, 2). Finally, this information will be fed from the physical device to a cloud application that can handle the data analysis or to a controller for alarm intervention (Fig 3.2, 3).

3.2 Conceptual Designs and Feasibility Studies

When initially installing ROS natively on the Flexxedge we found that the compatible version was not the most supported by the current UR_Robot_Driver. Therefore, we decided to use a virtual platform to run ROS called Docker. Docker is a platform to run virtualized software in packages called containers. Our resolution to this was to create a Docker container that hosts Ubuntu 18.04 and the associated ROS version, melodic. Melodic offers the most stable version of the UR_Robot_Driver at the time of development making it our best choice. The other advantage of using Docker is that it provides the ability to hot swap containers depending on the systems being used. Our testing was designed to run on a Universal Robot, but since we hoped to develop a widely usable platform, we needed to be able to make our software easily adaptable. Using Docker would allow for packages to easily be swapped and installed on the Flexxedge. An early example of our architecture without the use of Docker is shown in red Fig 3.3.

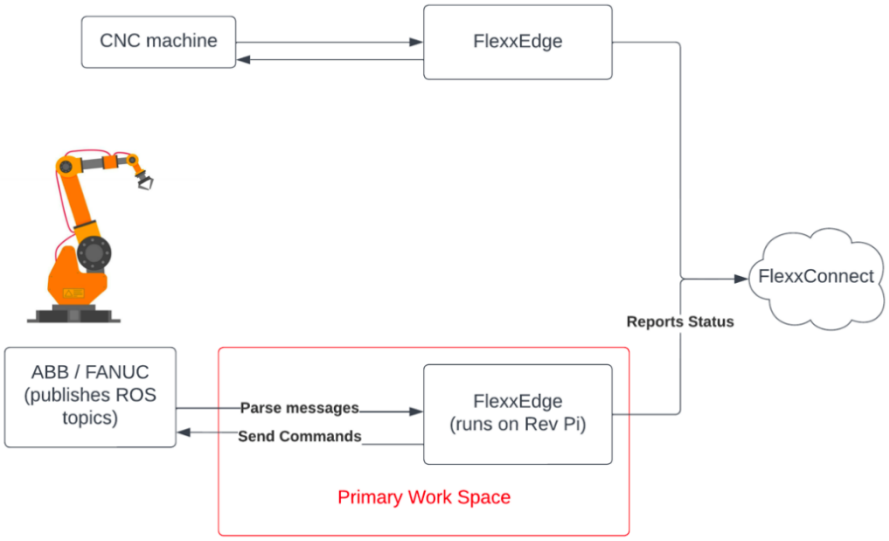

Figure 3.3 Workcell monitoring architecture

An adaptation of our proposed solution using ROS would be to implement ROS 2 instead. ROS 2, as the name implies, is a newer rework of the ROS architecture and has better performance and safety features. However, our initial tests revealed that the packages we wanted to use were not as well documented or built out as their ROS counterparts. When installing the software, alongside MoveIt and the other ROS industrial packages, we noticed a lot of features that hadn't been adapted or created that were relatively simple to implement and use with ROS. We assume that in the future ROS 2 will catch up with the original ROS, but for the purposes of our project we found the current state of ROS to be the best option.

Another large chunk of this project pertains to the actual CNC alarm detection. Our first attempt was to use DPRNT, a function of HAAS machines that allows the user to export data from the CNC machine as a job is running. The main issue with this option is that a job must be running for any data to be transmitted between the CNC and the monitoring device. This means that if an alarm is present prior to a job being started the monitoring device won't be able to see it, and the CNC would not allow a job to be started if an alarm is present. This prevents the ability to resolve alarm or do any form of alarm monitoring when the CNC machine is not running a job.

For production purposes there are software packages that offer a high level of CNC interfacing such as MT Connect, but for these were too expensive to be used in this work. There is however a status light that reports the state of the machine. When an alarm is present this changes the status light from green to red. This can be monitored via IO pins. When the red status pin is on, we know an alarm is present and this triggers the robot to generate an alarm report.

3.3 Core Functionality

The goal of this project was 3-fold: to detect errors that occur in our robot CNC work cell, to solve a select number of these errors, and to continuously record the status of work cell. Fig 3.4 demonstrates the typical workflow associated with our solution:

1. Normal operation: The arm tends the CNC by loading and unloading parts according to the job specifications loaded. Throughout operation, runtime information like part count and cycle time is sent to FlexxControl
2. Downtime detection: The work cell detects an alarm has been thrown by the CNC via IO and interrupts the normal operation
3. Alarm Report Generation: The tending robot presses the sequence of buttons to generate an alarm report to parse the details of the alarm
4. Alarm Resolution: The arm looks up the error in a resolution table and performs a series of actions to resolve this alarm
5. Cycle Resumption: The program is restarted and normal operation resumes

The technical details of each of these steps can be found in the following sections

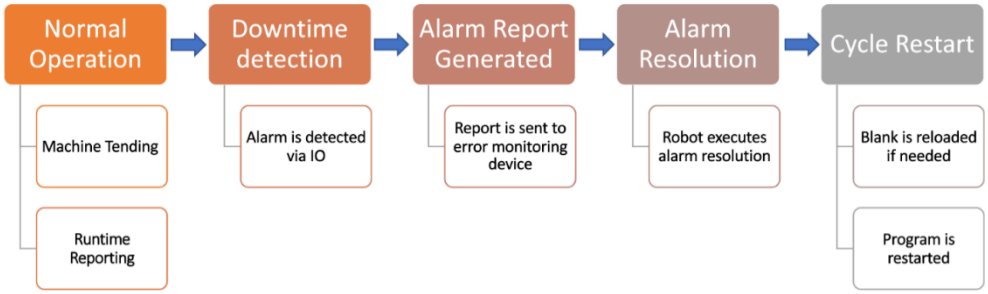

Figure 3.4: Workflow Diagram of the Work Cell

3.3.1 Alarm Detection

To be able to react to and resolve alarm codes, the work cell first had to be able detect when the CNC is down. The machine that we used does not have any way to directly record when downtime occurs, so an alternative method of detection had to be employed. When CNC machine is stopped or an alarm has occurred, the status light of the control console turns red. By connecting the I/O signal from the CNC that controls the light into the UR controller, the signal can be visible inside of our control environment as a ROS topic. Once the system detects that an alarm has occurred the arm would be sent to press the shift+f3 buttons on the CNC control panel which is the hotkey combination to quickly generate an alarm report.

Figure 3.5: Haas CNC control panel

This report includes a text file history of all alarms and status changes the CNC has gone through, and it is saved directly to the CNC machine. In order to get the report from the CNC machine into the error handler service, the NetShare functionality of the HAAS CNC was used. NetShare allows for the internal storage of the CNC to be shared with a second computer that is networked together with the mill.

Next, the error handler service parses the received text file to determine which alarm was thrown to interrupt normal operation. Fig 3.4 below demonstrates a view of the file which includes the alarm code, the description, and the timestamp for when the alarm occurred. Since multiple alarms can occur consecutively it was important to log not just the most recent alarm, but all the alarms that have happened recently. As such, the file parsing script creates a list of every alarm code that has been sent in the last two minutes to account for the time it takes to generate the report. The alarm file itself is just a text file. Data about the present alarms is extracted using a python script that can pull data based on the current date. If it sees an alarm that occurred within a certain time span, it then knows that this is the alarm that is currently stopping operation. There are occurrences where multiple alarms could be present and if this is the case the robot will perform both operations to resolve any present alarm.

```
206     ERROR READING PROGRAM                   2022/10/20 09:07:19 Device: N\A Prog Offset: 602 Program: USB0/mill_plate.nc
107     EMERGENCY STOP                          2022/10/20 09:23:29 Device: MDI Prog Offset: 0 Program:
160     LOW INCOMING AC LINE VOLTAGE            2022/10/20 09:23:34 Device: MDI Prog Offset: 0 Program:
20009   20009                                   2022/10/20 09:23:35 Device: N\A Prog Offset: N\A Program: N\A
107     EMERGENCY STOP                          2022/10/21 01:37:10 Device: N\A Prog Offset: 0 Program:
903     CNC MACHINE POWERED UP                  2022/10/21 01:37:11 Device: N\A Prog Offset: 0 Program:
102     SERVOS TURNED OFF                       2022/10/21 01:37:11 Device: N\A Prog Offset: 0 Program:
107     EMERGENCY STOP                          2022/10/21 06:50:18 Device: MEM Prog Offset: 0 Program: Memory/O02020.nc
20009   20009                                   2022/10/21 07:03:02 Device: N\A Prog Offset: N\A Program: N\A
903     CNC MACHINE POWERED UP                  2022/10/21 07:08:25 Device: MEM Prog Offset: 0 Program:
107     EMERGENCY STOP                          2022/10/21 07:08:27 Device: MEM Prog Offset: 0 Program:
102     SERVOS TURNED OFF                       2022/10/21 07:08:27 Device: MEM Prog Offset: 0 Program:
```

Figure 3.6: Screenshot from alarm history text file

After the list of alarms that need to be resolved had been created, it is time for the arm to perform the steps required to reset the CNC and resume normal operation. From the list of possible alarms, we ranked each alarm in terms of difficulty from one to three with three being the easiest. We grouped these codes together into alarm resolution “buckets”. This refers to codes that happen for different reasons but share the same resolution. For example, a significant number of resolvable alarms simply require the operator to press the reset button and then resume operation which constitutes the left-most column of alarm codes. By grouping

common alarms together, the path planning became much simpler to implement. Instead of each alarm having its own set of motion instructions we could limit the number of paths to only the number of groupings. Table 3.1 shows the set of buckets that we wanted to be able to resign with our alarm handler service.

Reset Machine	Reset ATC	Change Tool	Reload Part	Reload Program	Close Door
102	694	256	808	961	268
103	695	984	810		
104	696	994			
105	697				
292	698				
343					
176					
177					
971					

Table 3.1: Alarm buckets

3.3.2 Alarm Resolution

Once the work cell reliably detects errors in the CNC, the arm can then work to resolve these errors. These resolutions require motion between waypoints in the CNC workspace and on the controller, so we decided to implement the most common motion planning framework for ROS: MoveIt. MoveIt is flexible enough to be implemented with most other industrial arms and with different kinds of motion planning algorithms. Our main goal with any motion planning framework was to be able to send the robot to waypoints and avoid collisions at the same time. The package also provides the MoveIt commander class which allowed us to create way points in the form of ROS geometry messages and pass them into a go-to-pose function. This allowed us the flexibility to control the robot without having to perform kinematic calculations to get joint angles or avoid collisions. MoveIt also provides a visualization of the paths via RViz allowing us to test paths prior to running them on the physical robot.

To control the arm, MoveIt takes poses from either the joint or task space and passes them to a motion planner. To get these poses we used the lead through functionality of the arm to place it at desired poses and then recorded the joint and positional information at that location. After creating a list of necessary poses, we were able to create methods for each specific function, including pushing buttons and opening or closing the door. The motion planner then takes this goal pose and calculates how to move each joint to get the robot to the desired pose. Due to the nature of kinematics and 6 axis robotic arms there is not an all-purpose motion planner. Most are designed for a specific task or operation. While most motions could be planned in the joint space as the arm transitions between poses, we needed to be able to plan in cartesian space in order to close the door smoothly and push buttons accurately without incidentally contacting other parts of the CNC controller. Therefore, we used the PILZ motion planning pipeline that allows us to use linear, circular, and point to point motions to manipulate the arm.

MoveIt supports a variety of motion planners including the native planner OMPL and third-party planners such PILZ, CHOMP, and STOMP. (*Planners | MoveIt*, n.d.) We did some initial testing with OMPL and found it too unstable so we switched to PILZ. PILZ supported multi-motion smoothing and offered point-to-point and linear trajectory generation all of which were critical features to us. Additionally, it is a lightweight calculator that minimizes the computation time of path planning, and it generates complete paths before moving making it a safer option in a cramped workspace.

To make sure the arm reacts quickly to an error the job path is set inside of a while loop that immediately stops working when the alarm IO is detected. It then calls the error parsing file. This script returns any alarm that has occurred in the last 120 seconds. If the alarms present have resolutions that the arm can handle it will then perform the resolution. At the same time the Flexx system recognizes that a part has failed due to an alarm and counts it as a failed part that is then reported to the central system.

3.3.3 Runtime Recording

The final part of our project is to continuously record data about the work cell into the Flexxbotics environment. Flexx provides a solution that allows manufacturers to monitor multiple work cells on one system. A work cell can contain any number of machines, but the ones that use the Flexx solution normally contain a CNC machine and a robotic arm. Throughout operation of the CNC the work cell is monitored and all data it reported to a central system allowing for more efficient work floor management. The data type they use to record runtime information is called the run record. A run record includes the following information:

- Job name: identifier for the job currently running
- Part count: number of parts that have been created since the job started
- Robot status: the current status of the robot (RUNNING, STOPPED)
- Start time: the start time of the job
- End time: the time the last cycle ended
- Cycle time: duration of the last cycle

An example of this data can be seen in Fig 3.6.

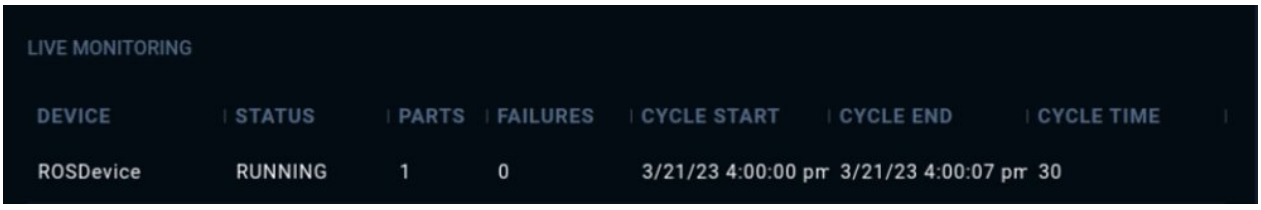

Figure 3.7: Run Record Example

While the ROS Industrial Interface greatly reduces the differences between implementing various vendor robots, it is not perfect. There are many topics that are vendor specific that require some specific implementation. Because of this specific implementation for some vendors, we decided to package the driver along with a set of translation nodes into a

single software image that could be easily shared and swapped out. For this project, Docker was chosen as the container service because it is extremely easy to get started with and was already implemented on the FlexxEdge. The components of the Docker image are shown in Fig 3.7 which include the ROS UR Robot Drivers and a set of specific ROS nodes that generate the run record information for the Hardware Abstraction Layer.

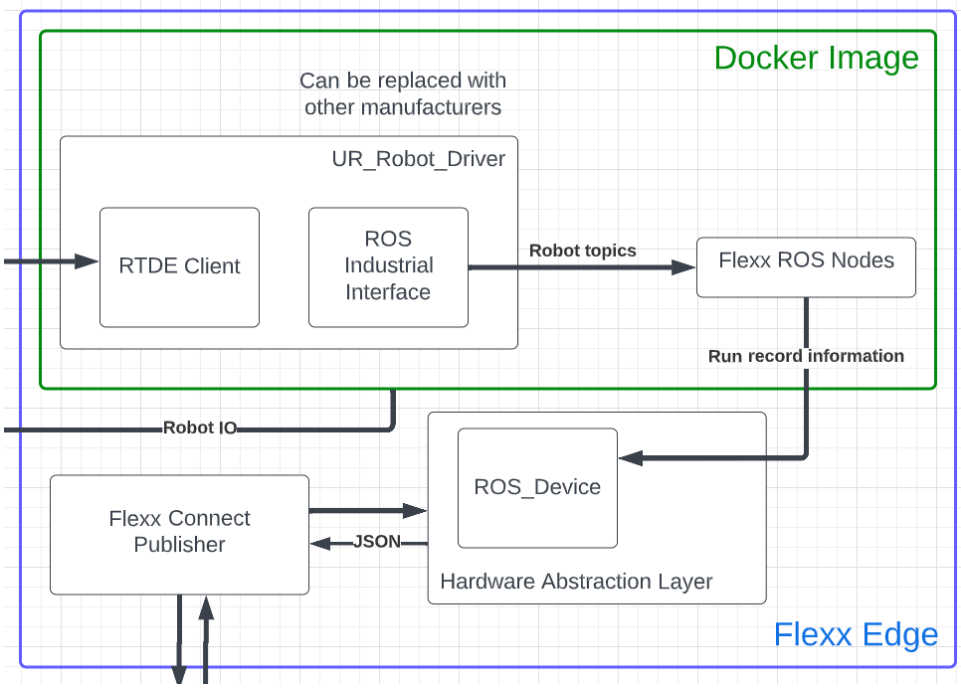

Figure 3.8: Docker Image within the FlexxEdge

The Flexx ROS Nodes referenced in the diagram are a set of nodes that calculate part count, cycle time, and failure count and then aggregates them to publish to the XMLRPC server. A more detailed visual of the nodes is shown in Fig 3.8. It primarily consists of 3 nodes:

1. The ur_robot_driver is the interface between the Universal Robot and ROS which publishes the hardware states and joint states of the robot.
2. The Runtime Generator node subscribes to the robot information to calculate the run record information like cycle time and part count.
3. The Runtime Aggregator publishes a complete run record at a set interval based on the components published by the Generator.

A full list of nodes and topics used in this project can be found in Appendix 1. While this is not completely generic as it still requires some extent of vendor-specific implementations, it is a great improvement over fully custom integration.

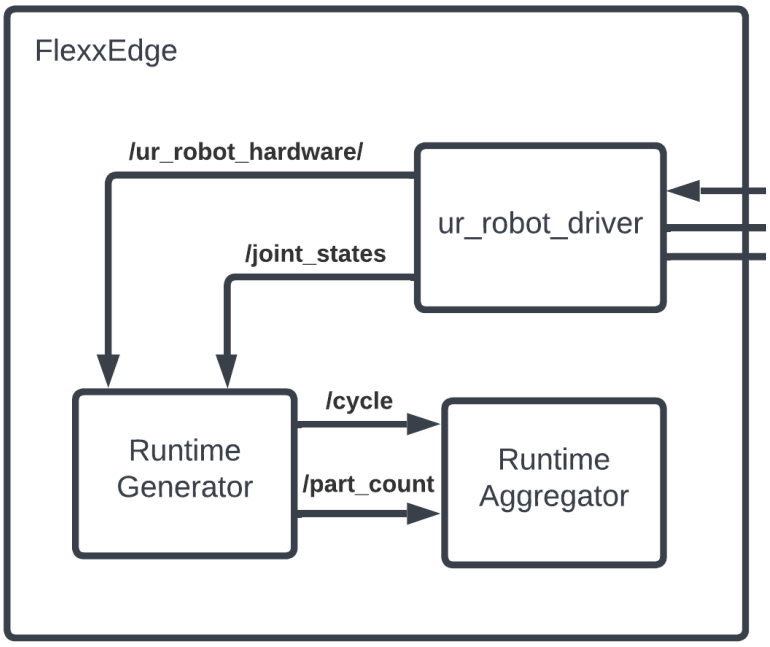

Figure 3.9: Nodes created to calculate and aggregate information for run records

In order to calculate part count, the user would define a pick position and a place position during the setup of the work cell. These positions correlated with where the robot grabs a new blank to put into the CNC machine and the position where it places the part into the machine. The Runtime Generator increments the part count every time the robot makes one complete cycle between these two positions.

The other piece of data that could not be accessed directly from the UR drivers is cycle time and overall job time. Cycle time refers to how long one part takes to be created whereas job time refers to the total time spent making a set of parts. For this the python time stamp package was used in the same node that calculates part count. When a job is started a time stamp is taken and saved; then when a job is ended another time stamp is taken. The difference between the time stamps is calculated and the result is the cycle duration. From here we could directly publish this to a job time topic. The same strategy was used for cycle time, but instead of taking the time stamps at the start and end of a job the stamp is taken whenever the part count is incremented.

Finally, to integrate our runtime recording with the Flexx environment, we had to add an abstraction to the Hardware Abstraction Layer that converts messages from the ROS device we created to a format their system can handle.

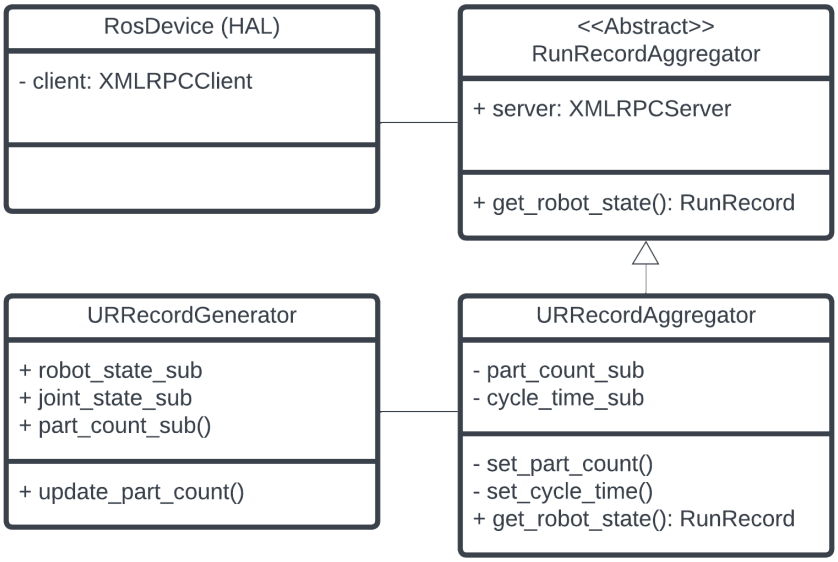

Figure 3.10: Class diagram of the Flexx ROS nodes that were developed

Figure 3.10 demonstrates the URRecordAggregator class that combines the robot state data into a single piece of run record information and exposes an XMLRPC server for the hardware abstraction layer to consume. The aggregator node subscribes to topics published from the URRecordGenerator which performs the calculations mentioned above. This implements an abstract RunRecordAggregator class so new generators can be created with different robot manufacturers without requiring full reimplementation. Finally, the HAL can get the robot state from the XMLRPC server and receive the current run record to push to the cloud and the web application for visualization as seen in Fig 3.10. shows the FlexxConnect web application is displaying the run record information that the HAL is printing out.

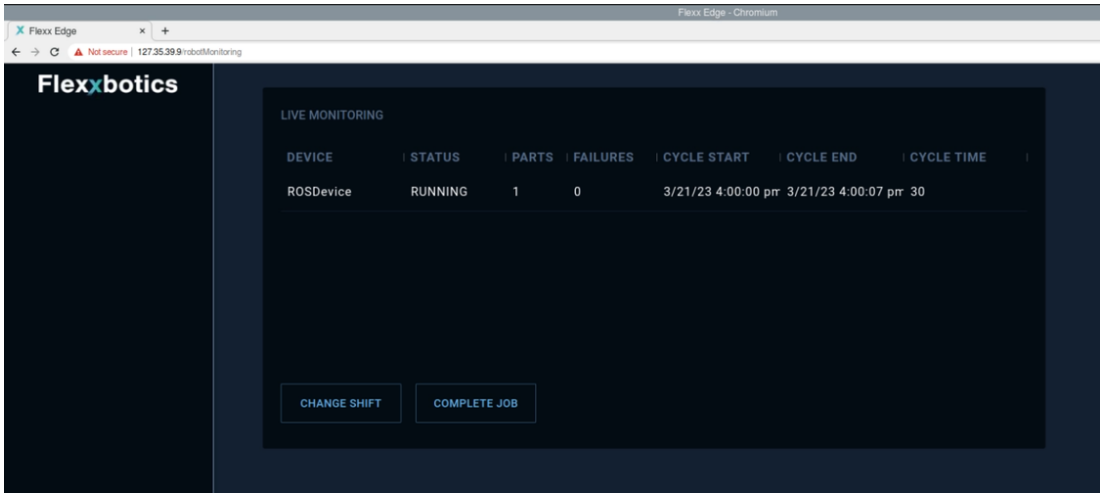

Figure 3.11: Visualizing the run record data in the web application

3.4 System Integration

Overall, integrating the system together worked well given the maturity of ROS and the flexibility of Docker. However, we had to deviate from our initial plans when connecting to the CNC machine. As described in 3.3.1, there are no native ways to access alarm information from the Haas Super MiniMill programmatically. This is why we had to work around this by having the arm physically push the buttons to generate an alarm report and analyze the text document to find the error code to resolve. This takes some time longer than being able to share the code directly with the robot. Additionally, the CNC networking functionality was never enabled so we had to prepare alarm reports before the trials for testing.

Secondarily, the motion planner had several issues when brought into the real world. The first is the number of dropped packets and how the planner reacts to them. Part of the reason PILZ was used is that it would not attempt to move through an unsafe path, and it had to generate a complete path at its initialization. However, this means if the planner tried to generate a motion while it had incorrect or partial information, such as from a dropped packet, it would fail to generate a path and end the program. To maintain the robustness of the motion planner while still being able to complete a whole cycle we had to implement a replanning protocol for the system. Since we knew the motions that we were sending the arm through were all viable from simulated testing our approach was to give the arm the same instructions at a delay if it failed when trying to plan a path. If the state of the robot prevented it from being able to identify a possible path initially, we only needed to wait until it had updated its kinematics and try again. To that end we allowed the arm to replan up to 3 times with a 0.1 second delay between attempts. This does not guarantee that the arm will never break, but for our purposes prevented the arm from having planning issues during our testing or demonstrations.

Chapter 4: System Testing and Validation

To confirm the efficacy of our approach we performed a series of tests. All the tests performed were performed on a simulated environment first in Rviz and then in the CNC workspace subsequently unless noted otherwise. The configuration of these environments will be listed in the subsequent section followed by the testing performed. The results of these tests will be used to analyze performance and determine whether the newly developed system was successful.

4.1 Testing Environments

Before any testing was done on physical hardware, we tested the system in simulation. The goal being to verify that positions were reachable and did not cause collisions. It also allowed us to test functionality for data aggregation when the physical robot was not present. The simulation used was Rviz the native ROS physical simulator. We recorded points that approximated the locations of the buttons, workpieces, and door. The arm was then sent through the positions in the order that would be necessary to complete tasks on the CNC without any feedback or signals from the mill.

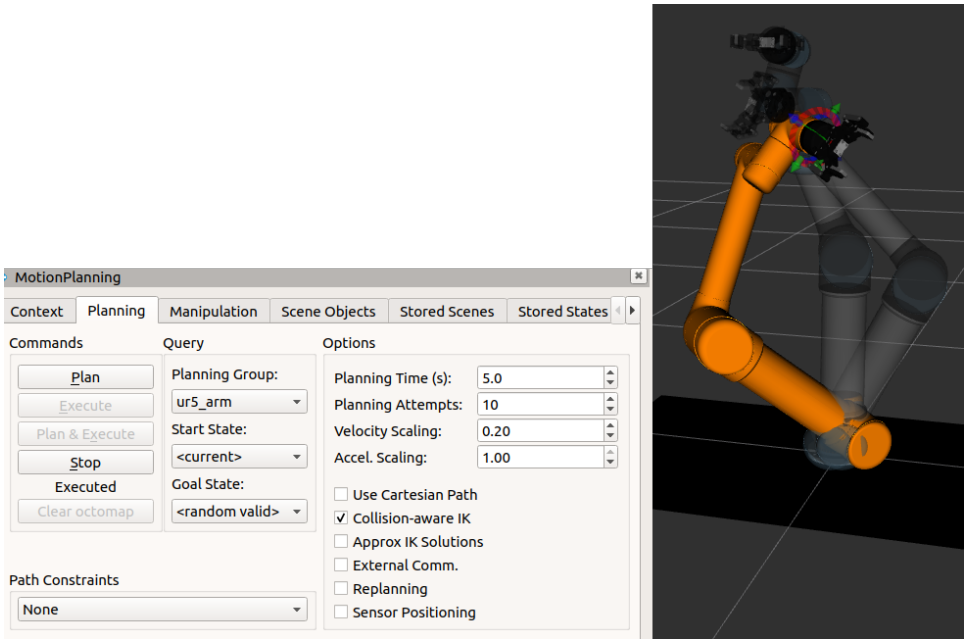

Figure 4.1: Rviz Simulation Environment

The transition from a simulated environment into using a real robot brought about several implementation issues that required slight changes to the testing. The most significant change was the addition of intermediate joint poses during motions to help assist the motion planner. To mitigate planning issues and avoid unsafe paths in the confined workspace several intermediate waypoints were added to the task that altered the job performance superficially.

Figure 4.2: CNC Workcell Configuration for Testing

An additional replanning functionality was necessary as discussed in section 3.4. Figure 4.3 shows the number of errors that occurred over the course of a single cycle. When we allowed the arm to replan three times it was anecdotally able to run without errors over the course of our testing. These changes brought the consistency of the physical system closer to the expected outcome from simulation although it added up to 0.3 seconds per motion of delay.

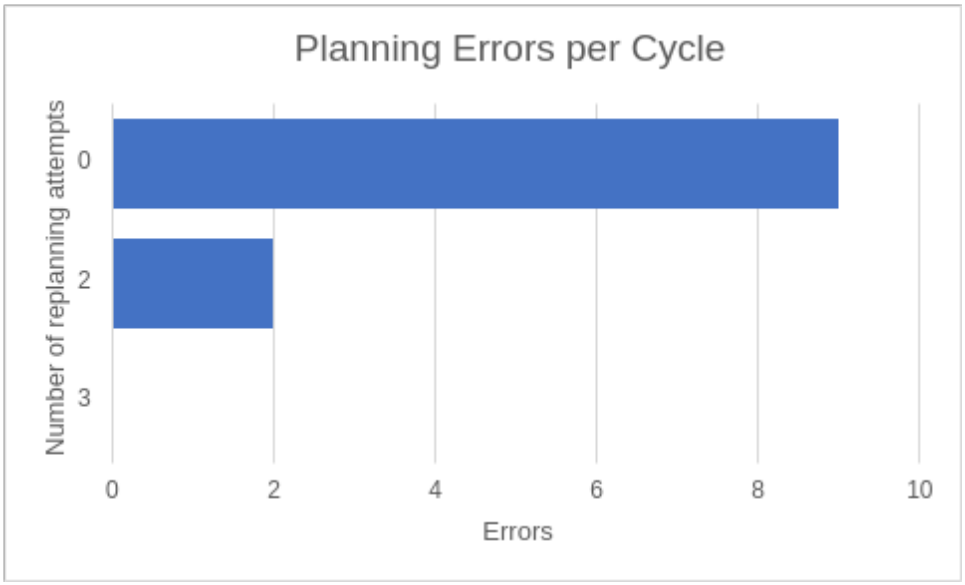

Figure 4.3: Replanning errors vs Attempts

4.2.1 Machine Tending Tests

The first step to testing the machine tending system was letting it perform unperturbed cycles in normal operation. First the motions were tested in simulation, then the lab without the CNC machine, and finally we moved to testing the ability of the arm to complete the tasks required to tend a CNC machine through the motions we had planned. To complete one job cycle the robot had to open the door, place the workpiece in the machine, close the door, start the cycle, wait for the job to complete, remove the completed workpiece, and then repeat the cycle again. Testing for the baseline machine tending task was simply to allow the arm to complete cycles and record the performance.

The testing for the alarm handler service was to allow the arm to allow the arm to perform a cycle and during the job to have a human disturb the system and create an alarm. For testing the ability of the arm to autonomously complete alarm resolutions we were limited in some respects. Haas provides a comprehensive list of alarm codes and a plain English description of the likely resolutions, but there is no way for the CNC to generate an alarm without the alarm being present in the system. Due to this we were only able to live test the alarms we could generate without damaging the machine, and the rest we had to artificially generate using a modified alarm history file. The other downside to artificially generating alarms is that the CNC is not in the alarmed state so we have no way to truly confirm whether or not our resolution would truly fix the issue associated with the given code. The final limitation on our test was that not every alarm sets off the safety light as we initially believed, so for specifically the open-door alarm code we would prompt the robot when there was an issue although it still had to parse the issue from the alarm history and react independently.

For live testing we were able to open the door during a cycle or turn the CNC machine off, and then see if the arm was able to return the machine to an operating state. For artificially generated alarms we developed individual tests that were close approximates or dry runs for what we believed the arm should do. Both ATC functionalities could be performed without an alarm present in the system so we would attempt to send the arm through the steps to dry run resetting the ATC and changing the tool. For the alarms in the “Reset Machine” column we used the emergency stop of the CNC machine which requires the machine to be reset before a cycle can be resumed. To test if the robot could hit reset during a cycle we would press and unpress the emergency stop button and prompt the robot with a modified alarm history file that listed one of the alarms that required a machine reset. For a corrupted program instruction, we would load a different program onto the CNC and then prompt the arm to reload the program.

4.2.2 Machine Tending Results

The arm was able to interface with the panel completely, and the UR5e’s workspace was more than sufficient to reach any part of the environment that would be involved in operating

the CNC. With the Robotiq two finger parallel gripper the arm was able to depress all the buttons on the panel, although the large, protruding buttons required a slight angle.

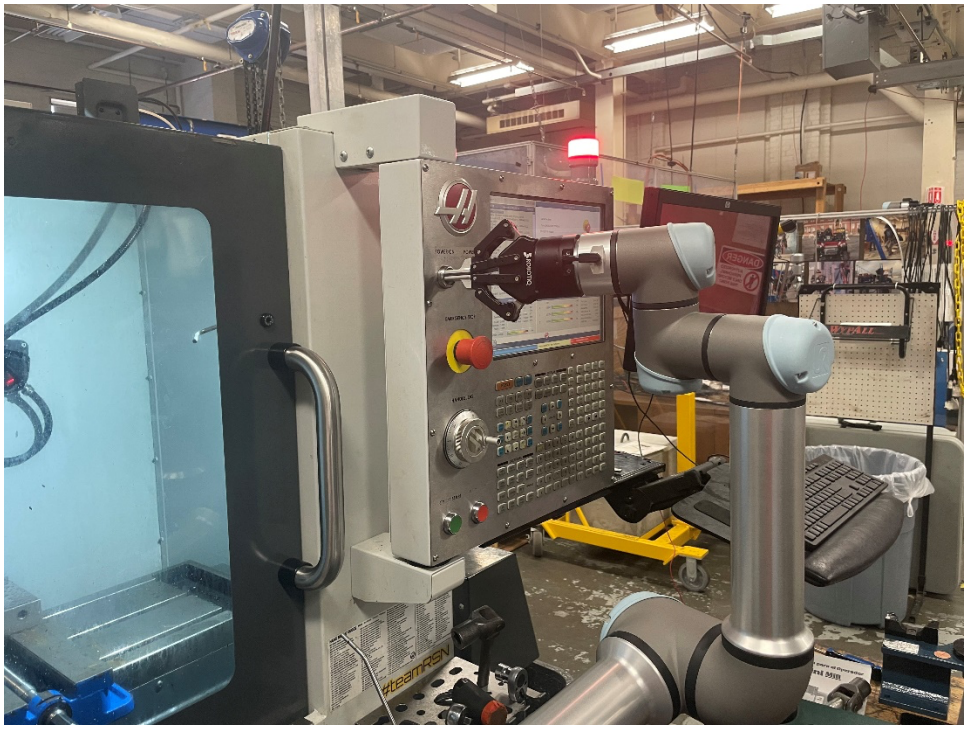

Figure 4.4: Arm interfacing with CNC power button

No long-term stress tests were able to be run due to limitations of CNC access, but the arm was tested up to five cycles consecutively and performed without issues or indications of potential failure. Over a two-hour testing period we were able to run the machine tending application without external assistance and saw no failures.

After testing the consistency of the machine tending application, we evaluated the speed of performance. While running the arm at 30% (limited for safety) of its possible speed we recorded the duration of the pre and post cycle steps and graphed the relative performance to a human in figure 4.4

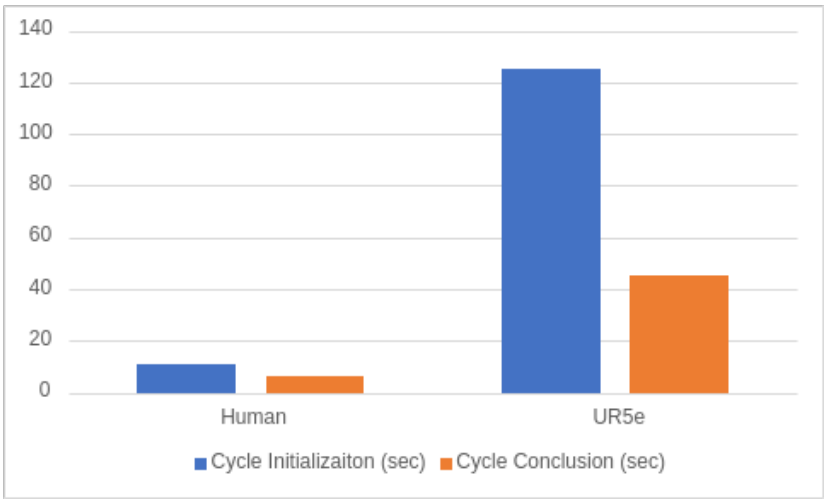

Figure 4.4 Cycle Times for Machine Tending Tasks

The next round of testing considered the ability of the alarm handler service to resolve alarms. Out of the categories, we were able to successfully demonstrate the system resolving Close Door and Cycle Power. These demonstrations can be found at the end of this section. Unfortunately, we were unable to safely throw the errors in the following categories: Reset Machine, Reload Part, or Reload Program. Therefore, only the solution motions were tested, but full integration runs were not. Finally, the tests for the Change Tool and Reset ATC category did not work as we found the Automatic Tool Changer (ATC) functionality of the Super Mini Mill required a greater depth of integration than our system was capable of. To use the ATC our system would have to have accurate knowledge of the tools, their positions, and their state which we were not able to access through the CNC. Because of this, the ability to Change Tool or Reset ATC was not considered to be viable since it would not be performed with the guarantee of safety.

Reset ATC	Change Tool	Reset Machine	Reload Part	Reload Program	Close Door	Cycle Power
694	256	102	808	961	268	20009
695	984	103	810			
696	994	104				
697		105				
698		292				
		343				
		176				
		177				
		971				

Table 4.1: Outcomes from alarm resolution testing

4.3.1 Runtime Recording Tests

Once we had developed a baseline of the workflow for what our machine tending task would look like we then had to test the runtime aggregation framework. The benefit of the RViz simulator is that the topics coming from ROS are published as though they were coming from a real robot. However, there are no noise or bandwidth issues to account for. The test for runtime recording was simply to run a job and make sure the FlexxConnect showed the correct information and that the run record had the correct information for the information that isn't shown on the UI as well. In simulation we did not consider runtime failures as part of the process, but we did consider QA failures. The rest of the components were considered normally for both simulation and hardware testing.

4.3.2 Runtime Recording Results

The data aggregation package was robust and consistent across the testing platforms. Even with the increased latency and noise the nodes ran identically in simulation and on real hardware. The addition of failure checking also worked without any major hiccups. Figure 3.7 shows the FlexxConnect webpage showing information collected via ROS and the conclusion of a complete monitoring pipeline. In simulation we were able to test the monitoring pipeline's ability over a long form testing period of eight hours. During this we completed 2000 cycles and despite one genuine job crash caused by an IP address bug the run record nodes continued to collect accurate information about the state of the job.

*Link to Demos: https://www.youtube.com/playlist?list=PLm7rP-S1QAZ9Bif7rpA7Ei2YUmYKntnF_*

Chapter 5: Analysis and Discussion

The primary goals of the system were all met, but there were additional criteria for how the system was developed. Firstly, flexibility was the primary goal for the structure of both the work cell monitoring and the error handler system. The benefit of using ROS for monitoring and control is that it can be used to run products from dozens of manufacturers, and we sought to capitalize on this. Secondly, since our system was designed to take the place of a human operator it was important to compare whether what we developed performed better than the human standard. Lastly, while the goal of this project was to develop new techniques primarily, we also wanted our solution to be as close to production level quality as possible, which means it would be designed and tested with the idea of it being used in a real work cell and that we could not compromise the consistency of our solution.

5.1 Flexibility

This project's primary goal was to develop a methodology that not only completed the tasks but did so in such a way that it could be implemented on a broad range of equipment manufacturers and easily expanded to include additional functionality. So, to evaluate the flexibility of our work we looked at the ease with which this approach could be reimplemented in other environments, and whether there are any limitations on the abilities of this approach.

First, we can evaluate the flexibility of the machine tending application and alarm handler. For this size of mill, it is important to avoid having too large of an arm as it will have difficulties operating quickly in a tight environment without damaging or hitting objects in the workspace, but if the arm is too small it would not be able to perform the full range of functionalities that the CNC has available. The UR5e arm range proved to be ideal as its workspace included every element of the CNC machine that we needed to have access to. With the arm centered on the door we had access to the full range of buttons on the CNC control panel and could reach far enough into the CNC to place the workpiece in the fixture and working area of the mill. It was also the robot both WPI and Flexxbotics had the most access to and experience with. This made it an easy choice due to availability and access to resources. In terms of motion planning there were very few poses that the arm was not able to reach, however getting to poses from every location proved difficult. Since the arm had to pass near the base joint singularity when moving from interfacing with the door to interfacing with the control panel depending on when an error occurred the arm could potentially try to move through an unsafe motion trying to react and find itself in an unsolvable situation. We did combat this to some degree by having each motion bookended by poses that gave the arm better access to its full range of motion, but this comes with no guarantee.

The error handler service itself was not designed to be fully general as it was simply a utility for our machine tending application, but apart from our alarm parsing approach it could be easily updated to include other manufacturers. The core functionality of the alarm handler service was to decide which resolution fit with the input given to it, and so long as the possible accepted inputs was expanded it could feasibly accommodate additional types of input. The

resolutions themselves were modular in nature. Core functions of the CNC were segmented such that the robot was deciding between processes and not poses. For example, the central loop of the machine tending was broken into several commands such as *turn_on()* or *open_door(),* and the alarm resolutions were structured similarly. To add new resolutions an engineer would only need to know what functions of the CNC were necessary and call those function commands in order.

Lastly the run record aggregation which had the most stringent requirements of generality. The methodology we used needed to be broadly applicable with minimal changes otherwise it would not suit the use case of our sponsors who needed a general use monitoring device. Through the existing open-source ROS packages we were able to develop ways that were hardware agnostic to get every piece of information except for running mode and controller IO values. These topics are not part of the ROS Industrial interface and are unable to be derived from other values. We expected some information of this kind to exist, so we designed the actual recording package itself to be easily editable and quickly swappable. The aggregation nodes are part of a docker image that can be downloaded onto the FlexxEdge that handles each specific manufacturer's differences. Updating the docker image to accommodate a new manufacturer is extremely lightweight and requires only changing the names of the two topics that are being provided via the manufacturer specific interface. With this an end user could have an image for each robot manufacturer and to change between them they would only need to run the monitoring nodes on the image designed for the device currently connected.

5.2 Relative Performance

The performance of the system relative to a human operator is mixed. The strong suits of automation are clearly at play in this system with the arm being able to move faster than humans at top speed and the arm never makes mistakes. The arm is also able to perform cycles continuously without breaks. However, the cycle speeds are not necessarily faster than a human due to the inefficient motions required to navigate the cramped workspace. Our pathing was not an optimal set of motions necessarily so with more time to hone the individual steps this methodology could outperform humans in speed, but ours does not. Additionally, since this is a cobot and not a traditional industrial robot the payload is limited to 5kg which is much less than the OSHA approved safe lifting load of approximately 23kg. While 5kg is a reasonable amount of weight for what can be expected for CNC material, if there happened to be a task that required a particularly heavy blank, the arm does not have the same capacity for lifting as a human. These limitations aside, the arm is able to perform expected tasks roughly at the same level as a human. However, the major drawback is that the arm still lacks the cognitive and observational strengths of a human operator. For example, if the door was opened because an object fell and it landed in such a way that it continued to be in the path of the door, the robot would have no way to know this.

5.3 Consistency

One of the primary claims of robotic arm manufacturers is that the behavior of their arms is highly repeatable. (Workers et al., n.d.) Without this they would not be useful tools for automation. Being able to perform things the same way is essential to developing an industrial application. The PILZ motion planner is consistent but not very robust to lost data. We were able to account for this by adding an additional layer of replanning that cut down the number of planning errors. This allowed us to perform the motions without failures for our testing purposes and this consistency during testing indicated a high level of reliability. Since PILZ is iterative there is a small degree of probability involved in path generation, but we were able to limit this to a negligible amount. When the functionality of the arm was expanded to include motions involving poses that were very difficult to plan paths for the likelihood of failures would increase but could be accounted for with the use of intermediate waypoints. Additionally, during testing specific points of failure could be identified anecdotally. Our replanner is parametric and if certain motions were identified as being more likely to fail, we were able to account for these and PILZ was given more leniency to regenerate viable paths.

Run record generation was extremely consistent and offered identical performance in simulation and on real hardware. Regardless of the goal positions or motions or length of cycle the aggregation nodes were able to accurately identify any calculated information and record any static information. Recording was tested both natively on the edge and hosted on an external laptop which also performed identically. Our run record reporting system was able to run live demonstrations for an entire workday without issue even if the job itself went down.

5.4 Lessons Learned

The first issue found with initial setup was inconsistent robot position from job to job. This led to inconsistent interactions between the robot and CNC or robot and part blanks. The best solution for this is something that already exists and is made by flexx as well. It's a reference in the workspace that has an unknown position in relation to the robot at the start of the job but all the necessary points for the robot to use are referenced to it. When a job is being set up the arm is physically moved to the reference point and told that its current position is the position of the reference therefore giving it the position of all the buttons and pick up points on the CNC. This is something that could have been done custom or done with the flexx device itself but due to time constraints it was easier to just adjust the position of the robot in the work cell by small amounts when testing.

Initial testing was done with the default motion planner for MoveIt. In simulation this planner worked great and had very little issues using all the move types. However, when the concept was proven in simulation and moved to the physical setup planning failed most of the time. After doing some research it was discovered that this is a well-known issue. The default MoveIt planner works great in simulation but has a hard time dealing with small errors in joint angles, acceleration, and velocity that generally occur in the physical setup. These errors could

be reduced by limiting movement speeds and forces, but this makes the overall system really slow. The better solution is to use a motion planner that is better suited for the physical robot.

The Pilz motion planner is a generic planner that is designed to work with a multi axis industrial robot arm. It supports point to point (PTP), Linear (Lin) and Circular (Cir) movements making it a suitable option for the tending and alarm resolution needs. However, there were still instances where the motion planner would fail. This mostly happened when the start and end point had very different orientations. The most practiced solution here is to create intermediate points that get the robot to some midpoint between the two crucial points. This made the number of planning failures a lot less, but they still happened from time to time. When a move failed it was discovered that if the move was retried from the point, it failed it worked without failure. Thes led to the idea of simply retrying the move in the event of a failure. Fortunately, when a move fails the planner throws an exception that when caught the plan is simply retried. If it fails, more than the predefined limit the planner with stop attempting to move the robot.

Chapter 6: Conclusions and Recommendations

6.1 ROS as a monitoring tool

This project showed that ROS can serve as an effective data aggregation tool even if the device is not using ROS natively. In the case of this project this was shown by the ability to pull alarm data from the CNC via an alarm list text file. One major key to the use of ROS for this application was the use of Docker. Docker makes the use of ROS much more convenient for use on different computer architectures. It allowed the solution to be much more generic and modular with the idea being that other robot drivers could be used in the future.

Another major advantage of using ROS as a monitoring tool is that many of the packages that expose the data pertaining to a CNC or robot operations already exist. The companies involved with ROS industrial have flushed out drivers that with slight alterations could be used to obtain even more data pertaining to the work cell.

6.2 Machine tending

Another fact that was displayed by this project is that the ability to do any form of machining with a robot/cobot is extremely valuable. When looking at the job itself it makes sense that a robot can succeed in the area. The job itself is repetitive, dull, and dirty making a robot the perfect solution for it, especially due to the fact that the environment that the robot is interacting with does not change during operation. This also made it very evident that the robot should stick to the material replacement side of the job, button pushing, door opening, and fixturing should stay in the hands of external packages that already exist. This, however, goes back to the point of ROS being a great bridge between different devices. Door openers and devices that can push buttons on a CNC in a simpler manner already exist, and ROS could serve as a better connection between all of them within a work cell and gain access to more data and metrics to report on the performance side of things. These devices were designed purely for the task they are completing, making them much faster than the robot, but limited in flexibility.

6.3 Performance

The optimal solution is the seamless integration of these two ideas into one CNC tending solution. Lights out manufacturing becomes much more feasible if the robot can run on its own without intervention from a human. This solution also serves as a great remote monitoring system by allowing companies to track multiple work cell outputs from anywhere in the world. This allows users to optimize work cell output by viewing machine metrics over time and seeing the areas that need to be improved. The other benefit of this is that again this is automatic. There is no manual data entry, the robot handles all aspects of the work cell operation. Using a docker image in the system as well makes the system's use with multiple robots manufactures in the same workspace much more viable.

6.4 Recommendations

In a perfect world a work cell would never throw an alarm and have a need for a resolution. Unfortunately, this is not the case and there is not a solution like resetting the

machine that can resolve any present alarms. Sometimes human intervention is necessary. The next best thing to autonomous resolution is the ability to control the robot remotely. Solutions like this already exist using camera feeds and controllers to allow the user to manually reset a machine from anywhere in the world. This idea could be applied to alarm resolution for alarms that require dynamic solutions. This would still allow for lights out manufacturing that is for the most part completely autonomous. Even if a teleoperated solution is needed the human is still not present at the machine.

Along with teleoperation comes the need to visualize the robot's current configuration remotely. This could come in the form of camera feeds or using the joint state ROS topic to get joint data and visualize it in a simulated environment. Either way this would provide valuable information to work cell managers that maybe isn't as apparent when just looking at metrics.

As shown earlier the benefits of using robots to handle these kinds of tasks cannot be understated, but the field would need to develop further to make it truly viable. The continued expansion of the stream of information allows for even more optimization in the workspace/work cells. Moving forward, to incorporate even more devices, all that would be required would be the development of packages that expose the required data for the monitoring device to access.

6.5 Future Expansions

As of right now the system was designed and tested on a CNC 3 axis machine, however it is generic enough that it could work with any other machine tending operation such as welding, turning and finishing. As mentioned, previously the system was also designed to be hot swapped for other robot manufactures. This design could be seen in a work cell that sands rocking chairs using a Fanuc arm, but still meets all the goals mentioned above. This of course was never tested for the duration of this project but was something that was baked in the overall design.

In the future a more robust solution to interface with the CNC machine could be used to speed up operation. Instead of reading the signal from the status light, software like MTConnect could be used to automatically report present alarms and statuses. An even better solution would be to design CNCs with the idea of using cobot in mind. Rather than using a bunch of what are essentially band aids to make the robot usable with a CNC, create a CNC that has built in communication protocols and IO’s to interface with the robot. Allow for the HMI (Human Machine Interface) to be controlled via IO rather than needing to physically press the buttons. These are all things that would make the interfacing between the robot and CNC machine much more seamless.

Glossary

Internet of Things (IoT) – How physical objects with sensors, processing ability, software, and other technologies that connect and exchange data with other devices and systems over the Internet or other communications networks

Work Cell - The arrangement of resources and processes, most often a combination of people, equipment, and materials, designed to improve the quality and speed of a particular output

FlexxEdge - Peripheral device that is networked into devices for recording performance analytics

FlexxConnect - Centralized hub application that packages performance data into visualization from an entire plant

Hardware Abstraction Layer (HAL) - A service on the FlexxEdge that exposes physical socket connections as publish/subscribe and request/response systems.

Run Record – Data type including status, failure count, part count, and cycle time from system devices that is used by the FlexxControl service

Robot Operating System (ROS): An open source software development kit for robotics applications. ROS offers a standard software platform to developers across industries (*ROS: Why ROS?*, n.d.)

References

admin. (2016, August 16). *Lean Manufacturing and Workcells: What You Need to Know*. https://online.kettering.edu/news/2016/08/15/lean-manufacturing-and-workcells-what-you-need-know

Al-Hussaini, S., Thakar, S., Kim, H., Rajendran, P., Shah, B. C., Marvel, J. A., & Gupta, S. K. (2020). *Human-Supervised Semi-Autonomous Mobile Manipulators for Safely and Efficiently Executing Machine Tending Tasks* (arXiv:2010.04899). arXiv. https://doi.org/10.48550/arXiv.2010.04899

Alli, B. O. (2008). *Fundamental principles of occupational health and safety* (2nd ed). International Labour Office.

Annem, V., Rajendran, P., Thakar, S., & Gupta, S. (2019, June). *Towards Remote Teleoperation of a Semi-Autonomous Mobile Manipulator System in Machine Tending Tasks*. ASME 2019 14th International Manufacturing Science and Engineering. https://www.researchgate.net/publication/337586346_Towards_Remote_Teleoperation_of_a_Semi-Autonomous_Mobile_Manipulator_System_in_Machine_Tending_Tasks

Arrais, R., Veiga, G., Ribeiro, T. T., Oliveira, D., Fernandes, R., Conceição, A. G. S., & Farias, P. C. M. A. (2019). Application of the Open Scalable Production System to Machine Tending of Additive Manufacturing Operations by a Mobile Manipulator. In P. Moura Oliveira, P. Novais, & L. P. Reis (Eds.), *Progress in Artificial Intelligence* (pp. 345–356). Springer International Publishing. https://doi.org/10.1007/978-3-030-30244-3_29

Borkhataria, C. (2017, July 5). *Radical "brain mesh" could make the Matrix a reality*. Mail Online. http://www.dailymail.co.uk/~/article-4668722/index.html

Caribbean, I. O. for L. A. and the. (2004). *2003 Labour Overview* [Report]. http://www.ilo.org/americas/publicaciones/WCMS_187481/lang--en/index.htm

CNC Machine Monitoring & OEE Software. (n.d.). Scytec DataXchange. Retrieved February 20, 2023, from https://scytec.com/

deSpautz, J. (1994). Quantifying the benefits of automation. *ISA Transactions*, *33*(2), 107–112. https://doi.org/10.1016/0019-0578(94)90041-8

Duong, L. N. K., Al-Fadhli, M., Jagtap, S., Bader, F., Martindale, W., Swainson, M., & Paoli, A. (2020). A review of robotics and autonomous systems in the food industry: From the supply chains perspective. *Trends in Food Science & Technology*, *106*, 355–364. https://doi.org/10.1016/j.tifs.2020.10.028

Flexxbotics | Manufacturing & Automation Productivity Solutions. (n.d.). Flexxbotics. Retrieved February 20, 2023, from https://flexxbotics.com/

FlexxCNC™ - UR to CNC Communication Interface. (n.d.). Flexxbotics. Retrieved February 20, 2023, from https://flexxbotics.com/flexx-cnc/

Hämäläinen, P., Takala, J., & Saarela, K. L. (2006). Global estimates of occupational accidents. *Safety Science*, *44*(2), 137–156. https://doi.org/10.1016/j.ssci.2005.08.017

HelmanCNC. (2013, July 9). *Haas Alarm Codes*. Helman CNC. https://www.helmancnc.com/haas-alarm-codes/

Jia, F., Jebelli, A., Ma, Y., & Ahmad, R. (2022). An Intelligent Manufacturing Approach Based on a Novel Deep Learning Method for Automatic Machine and Working Status Recognition. *Applied Sciences*, *12*(11), 5697. https://doi.org/10.3390/app12115697

Jia, F., Ma, Y., & Ahmad, R. (2021). Vision-Based Associative Robotic Recognition of Working Status in Autonomous Manufacturing Environment. *Procedia CIRP*, *104*, 1535–1540. https://doi.org/10.1016/j.procir.2021.11.259

Jia, F., Tzintzun, J., & Ahmad, R. (2020). An Improved Robot Path Planning Algorithm for a Novel Self-adapting Intelligent Machine Tending Robotic System. In E. E. Hernandez, S. Keshtkar, & S. I. Valdez (Eds.), *Industrial and Robotic Systems* (pp. 53–64). Springer International Publishing. https://doi.org/10.1007/978-3-030-45402-9_7

Katana, T. (2019, December 3). *The Benefits of Manufacturing Process Automation*. Katana. https://katanamrp.com/blog/manufacturing-process-automation/

Literature Review. (n.d.). Google Docs. Retrieved November 16, 2021, from https://docs.google.com/document/d/1u_skC48daew3YUoR48Mfb8Ov6Sw9WoBk9tgopqacr_Q/edit?usp=embed_facebook

Machine Tending Solution. (n.d.). Robotiq. Retrieved February 20, 2023, from https://robotiq.com/solutions/machine-tending

Martins, L., Varela, M. L. R., Fernandes, N. O., Carmo–Silva, S., & Machado, J. (2020). Literature review on autonomous production control methods. *Enterprise Information Systems*. https://www.tandfonline.com/doi/full/10.1080/17517575.2020.1731611

Michael, N. (2020, October 12). The Challenges of Robotic Perception. *Shield AI*. https://shield.ai/challenges-robotic-perception/

Mitchell, J. K., & University, U. N. (1996). *The long road to recovery :: community responses to industrial disaster /: edited by James K. Mitchell.* UN University Press,. https://digitallibrary.un.org/record/231666

Planners | MoveIt. (n.d.). Retrieved April 26, 2023, from https://moveit.ros.org/documentation/planners/

ROS: Home. (n.d.). Retrieved February 20, 2023, from https://www.ros.org/

ROS: Why ROS? (n.d.). Retrieved April 27, 2023, from https://www.ros.org/blog/why-ros/

ROS-Industrial. (2023, January 3). ROS-Industrial. https://rosindustrial.org

Save-to-transform as a catalyst for embracing digital disruption | Deloitte China | Strategy & Operations. (n.d.). Deloitte China. Retrieved March 2, 2023, from

https://www2.deloitte.com/cn/en/pages/strategy-operations/articles/2019-global-cost-survey.html

Statistics - Costs to Britain of workplace injuries and new cases of work-related ill health. (n.d.). Retrieved February 20, 2023, from https://www.hse.gov.uk/statistics/cost.htm

Syafrudin, M., Alfian, G., Fitriyani, N. L., & Rhee, J. (2018). Performance Analysis of IoT-Based Sensor, Big Data Processing, and Machine Learning Model for Real-Time Monitoring System in Automotive Manufacturing. *Sensors (Basel, Switzerland)*, *18*(9), 2946. https://doi.org/10.3390/s18092946

Tarapore, D., Christensen, A. L., & Timmis, J. (2017). Generic, scalable and decentralized fault detection for robot swarms. *PLOS ONE*, *12*(8), e0182058. https://doi.org/10.1371/journal.pone.0182058

Tarapore, D., Lima, P. U., Carneiro, J., & Christensen, A. L. (2015). To err is robotic, to tolerate immunological: fault detection in multirobot systems. *Bioinspiration & Biomimetics*, *10*(1), 016014. https://doi.org/10.1088/1748-3190/10/1/016014

The advantages of automated machine tending - Fanuc. (n.d.). Retrieved February 20, 2023, from https://www.fanuc.eu/dk/en/industrial-applications/machine-tending

The Cognitive Costs and Benefits of Automation. (n.d.). Retrieved February 20, 2023, from https://apps.dtic.mil/sti/citations/ADA422303

Workers, E. A. of K. P. T. C. to E. S. for, Production, A., & Productivity, I. (n.d.). *Industrial Robotic Arm Overview*. Intel. Retrieved April 27, 2023, from https://www.intel.com/content/www/us/en/robotics/robotic-arm.html

World Day for Safety and Health at Work 2009 - Facts on safety and health at work issues. (2009, April 27). [Fact sheet]. http://www.ilo.org/global/topics/safety-and-health-at-work/resources-library/publications/WCMS_105146/lang--en/index.htm

Appendix 1

This is a comprehensive list of all the ROS nodes used in this project which can be helpful for someone attempting to implement the system for a new work cell.

Alarm Resolution:

For the alarm resolution we created a ROS package called `ur5_gripper_moveit_config` which contained the following nodes:

1. CNC_Interfacing: the main job file that was used to run the error handler service which listened for errors and uses the Pilz Library to move the arm for resolution
    a. Subscribed to `/ur_hardware_interface/io_states` to read the errors
    b. Service Proxy to `/ur_hardware_interface/set_io` to open/close the gripper

Runtime Monitoring:

All the nodes for the runtime monitoring are hosted inside a Docker Image based on the official ROS image (ros:melodic-robot) with the Universal Robots ROS Driver already installed. The following nodes are hosted in the image:

1. URRecordGenerator: the node that listens to the specific UR topics to publish generic runtime information based on the run record schema
    a. Subscribed to `/joint_states`
    b. Publishes on `/part_count`
    c. Publishes on `/cycle_start_time`
    d. Publishes on `/cycle_end_time`
2. URRecordAggregator: the node that aggregates the information from the generator into a single run record and exposes it via RPC
    a. Subscribed to `/part_count`
    b. Subscribed to `/cycle_start_time`
    c. Subscribed to `/cycle_end_time`
    d. Subscribed to `/ur_hardware_interface/robot_mode`
    e. Subscribed to `/ur_hardware_interface/io_states`