API abstraction of Robotic Frameworks and its usability impact

Bachelor of Science Thesis in Software Engineering and Management

Enrique Cordero Miranda
Alexander Zajac

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

This paper explains the process to design and develop an API to abstract robotic frameworks away from users. It also evaluates the API in comparison to a robotic framework in regards to usability. In order to evaluate usability a systematic data collection process was done to measure operability, learnability and user error protection. The results show the API as a more usable point of contact than the robotic framework.

Supervisor: Pierguiseppe Mallozzi
Examiner: Christian Berger

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

API abstraction of Robotic Frameworks and its usability impact

Enrique Cordero
Computer Science and Engineering
University of Gothenburg
Gothenburg, Sweden

Alexander Zajac
Computer Science and Engineering
University of Gothenburg
Gothenburg, Sweden

***Abstract*—This paper explains the process to design and develop an API to abstract robotic frameworks away from users. It also evaluates the API in comparison to a robotic framework in regards to usability. In order to evaluate usability a systematic data collection process was done to measure operability, learnability and user error protection. The results show the API as a more usable point of contact than the robotic framework.**

***Index Terms*—REST API, Robotics, ROS, Opendavinci**

I. Introduction

A. Background

The improvement in technology and cheaper access to hardware has led to an increase in the use of technology to solve daily tasks. Current trends like the Internet of Things (IoT) [1] and Cloud Computing [2] [3] aim to digitize a lot of the comforts people have in their homes. The rise in popularity of the Internet of Things (IoT) [1] also impacts the use of robots today. Internet of things (IoT) is a common term currently used to relate to embedded systems, which are connected to the Internet to exchange data. This encourages robots to be more and more engaged in everyday home activities and require therefore some sort of automation and decision making capabilities based on their setting and their task [3] [1].

This has triggered an interest in decision making algorithms and artificial intelligence in the department of computer science and engineering at Gothenburg University. One of these researches focuses on self-adapting systems for robotics. Self-adaptive systems are systems that are able to adapt their behaviour at run-time without human intervention [4] [5] [6] in response to changes in the environment or in their internal state. A self-adaptive system gathers knowledge from the environment, contextualizes it considering also its internal state and adapts itself to achieve its goals. A well-known reference model for describing the adaptation processes is the MAPE-K loop [7]. Self adaptation can be achieved through machine learning.

The interest in different areas of robotics have triggered different challenges. The growth in everyday usage of robots has increased the number of frameworks available to handle the programming of such robots [8]. With such a broad availability of frameworks, programmers are faced with the challenge of having to learn about the technical details and how to communicate with them. All the different type of frameworks and robots make the term of ubiquitous computing to appear as a challenge with interconnecting all robots and exchange information in order to create a system which is more context aware and location aware [9]. Also, in order to validate the performance of machine learning algorithms on different platforms some technical challenges need to be addressed.

B. Problem Domain and Motivation

The above mentioned challenges develop into specific problems. Currently applications are usually programmed to communicate with a specific framework, which limits the code to be used again with other frameworks due to possible compatibility issues. With robotic software becoming complicated, it is vital to find an option to develop reusable robotic software [10]. Also robot systems are heterogeneous systems which change in a fast pace. We need a standardized system of communication, which web-services offer in a rather mature and developed platform.

Robots as web-services help avoid several limitations that current robot solutions have today which usually makes them language specific, or at least device specific [2]. Finally, to be able to test the same machine learning algorithm in different frameworks, abstracting the perception component and the execution with high level API common to the frameworks is necessary. Different existing solutions are framework specific, and therefore we have chosen to create an API. Creating an API to simplify the interaction with the frameworks will increase usability for the users and solve the aforementioned problems. We have chosen protobuf as the standard package to be exchanged because protobuf enables for less boilerplate code, and allows for backward compatibility, making the API more stable to version updates. It also allows for better performance since it does not parse sequentially bit by bit like JSON does. It parses the package depending on the package type and the structure bytewise. We have chose to implement the methods getCamera, setSpeed and setAngle to cover the entirety of movement and camera feed needed to choose the movement. These basic commands allow for image evaluation and movement once the decision has been made. Furthermore exposing this API through REST interface would free the developer to access the source code in order to use the API and also would enable other IoT scenarios to communicate with the robot remotely and to have access to the methods through the cloud and therefore providing a restful webservice.

A literary review has noted a gap in previous work were similar solutions have been proposed without a mention of whether or not the solution is actually more usable. This study will therefore evaluate the API by its usability. The usability of the API will be evaluated upon three specific variables, which are learnability, operability and user error protection. The other characteristics of usability defined by the ISO 25010 [11] are out of scope for this paper. The objectives of our research are stated as follows:

- Build the API
- Collect Data comparing the API to ROS as a representation of one of the frameworks
- Analyze the Data to show an increase in usability

C. Research Goal and Research Questions

The objective of this paper is to explain the design and implementation of a restful API, to abstract robotic frameworks and to evaluate the impact upon usability of having a single point of contact (API) to communicate with a robot. The main objective of the API is to improve usability by providing a single point of contact with an established protocol for applications to interact with the robotic frameworks. The paper intends to address the following research questions:

1) Main Research Question:

- RQ1: What is the impact on learnability, operability and user error protection of having a single point of contact (API) to a robotics system contra direct contact to the framework itself.

2) Sub-Questions:

- RQ1.1: Can a group of students, already familiar with robotic frameworks, make a simple application using the API without prior knowledge of the API?
- RQ1.2: What impact will programming experience have when evaluating operability, learnability and user error protection of the API in comparison to the robotic framework?

D. Hypothesis

Our hypothesis is that:

- H1: There will be a significant difference between having one point of contact to a robotics system contra direct contact to the framework itself.
- H0: There will not be a significant difference between having one point of contact to a robotics system contra direct contact to the framework itself.

The significant difference for the point of contact will be evaluated upon the three usability aspects mentioned above and will be considered to be significantly different if the majority of them have a significant difference.

E. Contributions

The scientific contribution is to conduct an empirical study to evaluate the impact an API would have on the usability of programming an application for a robot in comparison to doing so directly with the robotics framework. More specifically this paper will evaluate learnability, operability and user error protection as per defined by the ISO 25010 [11]. It will also be contributing to the ongoing research at Gothenburg University regarding self-adapting systems. The technical contribution is an API that provides the possibility to validate the performance of machine learning algorithms on different platforms.

F. Scope and Delineation

The usability of the API is evaluated against the ROS framework upon three specific variables, which are learnability, operability and user error protection. As mentioned before the other characteristics of usability defined by the ISO 25010 [11] are out of scope for this paper. This study will be addressing the usability of both ROS and and API within an educational context. It will be evaluation the movement and camera capabilities of the robots in the attempt to address a homogeneous robot. The rest of the possible capabilities of ROS and robots in general are out of scope.

G. Structure of the Paper

The rest of this paper is divided as follows: Section II describes the related work and more specifically the work already done behind service oriented architectures and robots as web-services. Section III defines the background behind the API and explains the architecture behind the system and the API itself. The methodology chosen for the paper is described in Section IV where you will also find the data collection and the evaluation techniques. The results are shown in Section V and a more elaborate discussion on the results is continued on Section VI. Finally section VII presents the conclusions and the possible future work to be done on the topic.

II. Related Work

Service Oriented Architecture (SOA) has also become a popular topic considering the rise in popularity of the IoT. Yinong et al. [2] reference the concept of SOA to strengthen the field of robotics by trying to apply the concept and build what they call Robot as a Service (RaaS). The main idea is to expose the robotic services to the end user and the developers in order to have a layer of common services and standard interfaces. The authors also kept within the margins of the WEB 2.0 principles of participation to comply with common service standards. However at the evaluation stage of the paper they focus on the portability and the flexibility of the design. Through experiments they show how the software and hardware efficiency. They do not however evaluate the impact that this type of architecture would have on the usability for the end user and developers. Since the end user and developers are mentioned as the primary users of the framework then certainly their experience of using the robot should change.

Raas did lay the groundwork for upcoming researches to look at the SOA paradigm and try to focus on robots as services.

Doriya et al. [3] continue to exploit the SOA paradigm to present a model and a simulation of robotic in SOA and cloud computing. The authors propose a framework and use Microsoft Robotics Developer Studio (MRDS) to simulate and test the framework they previously proposed. Through their proposed framework and the simulator capabilities the authors were able to use voice recognition to provide some services in order to give an example of the possibilities of having robots as Web services. The paper has a theoretical approach to expose the topic as an interesting one and shows the possibility of robot virtualization with web services. However there is no technical implementation nor was there ever an intention of measuring whether it would have any impact on the developers or the user.

Osentoski et al. [12] proposed to expose the services of the ROS framework to the users through rosjs and rosbridge. The motivation in their paper to create these tools was to be able to use a tool that is common for everyone as is a web browser and to give everyone a chance to program for or interact with robots. The main idea is to have a middleware called rosbridge to interact with ROS and pass the information along. On the web browser there is a javascript library called rosjs which would be able to receive this information and display it on the web browser. They also use JSON serialized messages to interact with the web interface which will encode / decode to ROS serialized messages to interact with the ROS back end. The biggest milestone this paper covers is the usage of more popular tools like javascript and JSON messaging. Also they provided alongside the possibility to connect with non-Web clients based on sockets (version 1) and websockets (version 2). As mentioned before this paper laid a strong practical foundation using a well known framework like ROS and established WEB tools like javascript and JSON. However they did not aim to evaluate the impact the tools they had created had on the developers. Like Yinong et al. these authors aim at creating something that would aim to help developers communicate through the web with a robot in order to facilitate the creation of applications for them, yet failed to evaluate if there was any help at all by the change.

Kim et al. [9] employ Web services to allow a robot to access a set of distributed services. The authors discuss the importance of ubiquitous functions and they explain the fundamental ubiquitous functions management framework for a robot. The paper also shows how the different layers and services would interact as a system to provide the required services to fulfill a task. However the paper does not mention the impact this would have on the users or on the developers. It has abstracted any specific robotic framework away from the user and created a generic solution in which the services are modular and independent. The authors have proven therefore in their paper that web services can be a solution to the problem of integrating different robotic solutions. The aim of this paper is to go a step further and try to evaluate the impact the tool will have on third party developers and users.

In his previous work Koubaa [13] tries to exploit the advantages of the SOA paradigm to expose robotic resources as services through the Web. The main idea behind the project was to allow non-technical user to manipulate robots through a Web browser. The author created a SOAP based middleware that exports the topics from the ROS framework to the user and therefore allows him to manipulate the robot through the Web browser. The author mentions the aim of the project is to allow non-technical users to manipulate robots. He even mentions how a possible experiment might be conducted however he never mentions what would be the goal of the experiment nor does he mention in the paper if he achieves in allowing non-technical users to manipulate the robots. Therefore we see again the need to evaluate whether the work done to expose the services of the robot would actually have an impact on the end user and / or the developers.

More recently Koubaa [14] has developed a more thorough web-service possibility for ROS. The author relies on the boom of the IoT and the complexity of developing client applications for robots as a foundation of why he chose to develop such a solution. The author chose to integrate web-service solution into the ROS framework to provide both REST and SOAP solution for developers to rely on. The author's experimentation validates their desire to promote portability, reusability and interoperability between ROS robots and client applications. Koubaa however aimed to make it easier for developers to interact with robots and he has not tried to prove just that. His work was completely aimed at developing the solutions and evaluating if the code could be more portable and reusable. The usability of the system he proposed in comparison to the ROS framework directly has not been evaluated.

III. API Background

As mentioned before the reason to create the API is to assist the current ongoing research at Gothenburg University regarding self-adapting systems. The research is centered around ROS and OpenDaVinci as the robotic frameworks to be used for the evaluation. OpenDaVinci [15] is a realtime-capable robotic framework written in standard C++. Currently the framework is platform specific. Currently it offers communication features such as UDP, TCP, shared memory and serial port. It offers the possibility of time-triggered or data-triggered software modules. ROS [16] is a flexible robotics framework aimed to simplify the creation of software for robots. It has a wide variety of libraries, tools and conventions to help in the creation of such software. It also aims to encourage the collaboration of different teams to develop and share the information with each other through releases of new libraries. ROS is based on the publish/subscribe pattern to allow different modules to communicate with each other. It is considered the main framework for prototyping and development of robotic software [14].

The idea behind the API is that it will allow test algorithms to run on different frameworks without the need to change the source code. Similar to how a webpage can be rendered the same way regardless of the web-browser used [17]. To be able to control all mobile robots with the same code it is needed to be able to control homogeneous robots. This considered, every framework has done a good job with adapting to the specific robot capabilities they intend to code for. Therefore it is not the aim to try to establish a new framework but instead push for a communications module inside each framework. This module would communicate with a central communications module which is referred to as the API. This way robot functionality can be provided as a web service. Robots as web-services help avoid several limitations that current robot solutions have today which usually makes them language and/or device specific [2]. The proposed architecture of the whole system can be seen in figure 1. The figure shows both the ROS and Opendavinci frameworks connecting to the REST interface. For the sake of this paper focus will only be on the ROS framework, the ROS connector and the REST interface.

The internal workings of the API is based on three basic methods to control the robot and to be able to evaluate its position. The three methods considered are getImage, setSpeed and setAngle. The idea is to be able to lay the grounds for basic movement and position evaluation of robots with the future capability of adding other capabilities offered by the framework. The API counts with the possibility of being accessed locally as a library through the methods mentioned above. It also has the possibility of being accessed through a server remotely. It has two main components called movement master and image master which handle the communication of the components. Each master is in charge of initializing the subcomponents that will communicate. The subcomponents have been divided in a way that they do not need to be deployed in the same machine in case of a distributed system. Figure 2 shows a more detailed component diagram from the web server to the ROS framework. It is important to mention that the Rest Interface layer which holds the movement master and the image master can also be accessed locally on the same local area network (LAN) and does not require the web server.

IV. Methodology

Design science has been chosen as the research method given the need to develop the API in order to build upon the ongoing research regarding self adaptive systems. The aim is to develop an artifact and to test it against the ROS framework and extract both quantitative and qualitative information from the participants during our data collection. Given the nature of the problem and the need for iterations, the aim was to do a flexible study. It has grown incrementally and has a positivist point of view starting with the hypothesis of the API being more usable than using the ROS framework directly. The study is based on the paper written by Koubaa [14] given that it is the most extensive paper found and the fact that it covers every point aimed to be evaluated. A layer of evaluation has been added to it regarding usability. A REST API was designed to communicate with the ROS framework which allowed the researchers to collect data in an easier way. Therefore the study can be seen as an improvement study to try to complete previous work with a real life comparison between web service API vs direct contact with the framework. The evaluation of the artifact will be done through a case study given the need to evaluate the API in a real case scenario. The data collection will be of first degree given that we will be working directly with the participants to get the information. The researches have also prepared the goals in advance and have formulated the questions and metrics according to them. Therefore the study can be considered to be using Goal Question Metric method (GQM) for the data collection. At the end of the data collection there was a small survey performed to evaluate some basic information from the participants and their general perspective of the tool they used. Descriptive statistics were used for data analysis of the quantitative data such as time and number of errors. The evaluation of qualitative data was done through pre-defined rules to avoid bias and have an even evaluation of the data /citeRef 37. The evaluation process will be described in detail in section IV-B.

The artifact is a RESTful API with a standard protocol that allows any framework to connect to it. In this way the API abstracts the framework away from the programmers in order to simplify the coding of applications that could possibly interact with the robot. To do this python has been chosen as the programming language to design a library that would run alongside ROS and communicate with the already existing ROS framework. A server in python has also been designed which provides not only local RESTful services but also allows these services to be accessed remotely. According to Stefik and Siebert [18] Python is among the languages that had a better outcome in their study in comparison to C related languages based on usability for novices. Given that ROS supports both C and Python we chose Python as the most user friendly of them. This allows for a wider connectivity and interaction as aimed to fulfill the RESTful services and the possibility to interconnect robots in the IoT scheme. The API has been designed based on the client/server model [17] which brings advantages such as scalability, concurrency and language independence.

It was chosen to design an API instead of using an existing one given that the existing ones are broad and specific to a framework. The aim was to design an API that would be generic enough as to be able to be applied to any framework. The main existing API in consideration to use was the rosbridge API [12] given that it is already launched and has good tutorials. However this tool is extensive and specific to the ROS framework. The data collection is aimed at having an API which relies on web services and could be generic enough as to apply to a homogeneous robot. The aim is also to be able to generalize the API enough as to make it a part of upcoming trends as cloud services and the IoT and not only have it oriented at a specific framework like ROS. Therefore the design of the API is as generic as could be imagined for a robot with movements and the possibility for a camera feed. The finished product is therefore a basic use of HTTP REST services through the use of a Get and Post methods.

The data collection is devised to evaluate if the API makes it more usable for 3rd party developers to develop software that will communicate with a robot, rather than using the robotics framework to accomplish the same task. The data was

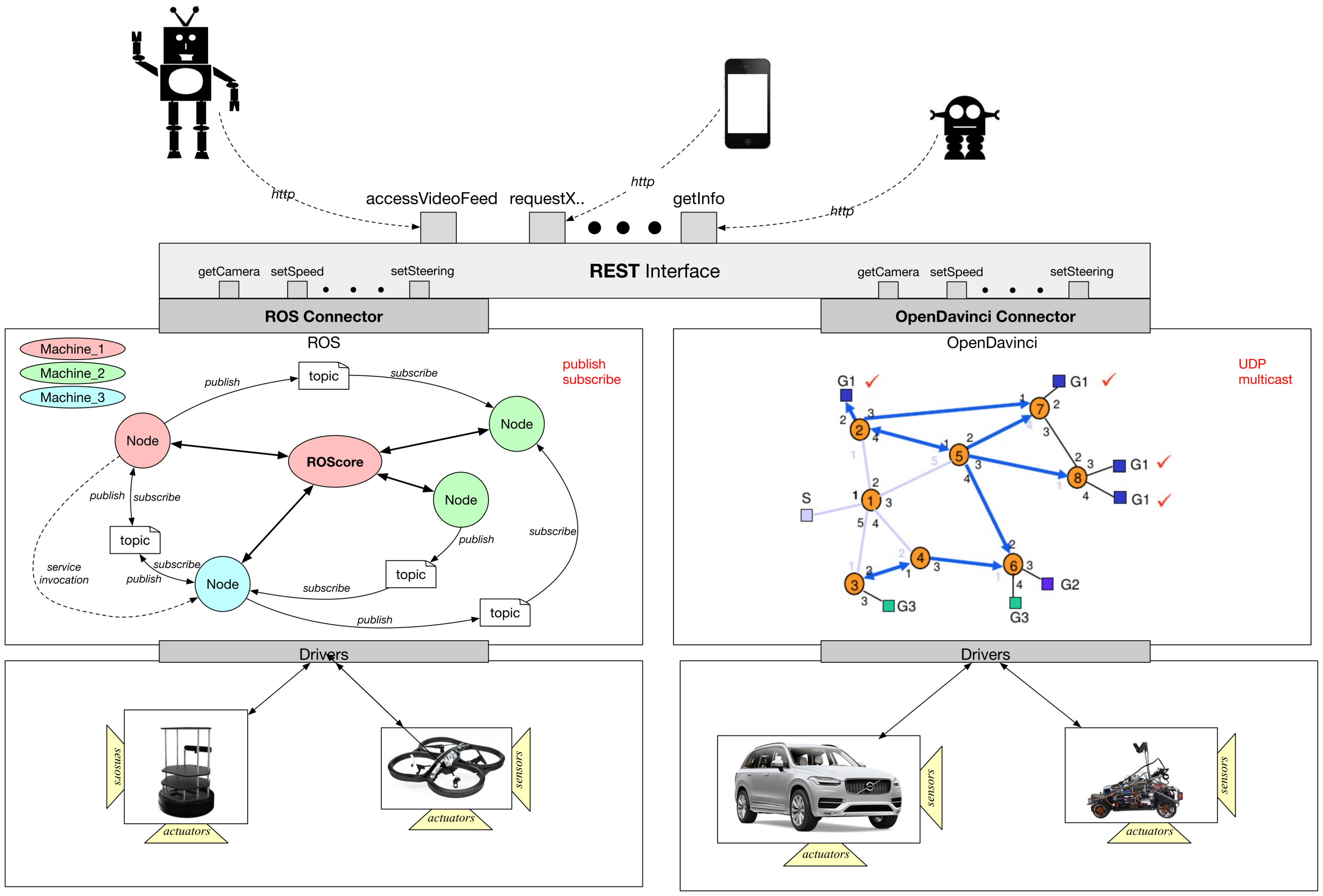


Fig. 1. System architecture including both ROS and Opendavinci.

collected both quantitatively and qualitatively from students at the University of Gothenburg who are in their 2nd or 3rd year of software engineering and management. The process consisted of 19 students who have or are currently working with the OpenDaVinci framework [15], which is a robotics framework for autonomous cars. Each student was given different tools to solve a series of challenges, some students were given the API functionality sheet and others the ROS framework functionality sheet. The challenge was constructed in such a way that students with no prior knowledge of either ROS or our API will be able to solve it, or parts of it in the given time. The parameters measured are:

- time to finish a task
- tasks that were completed correctly
- number of errors made during the challenge.

These parameters focus on measuring usability aspects of the API and the ROS framework. Usability will be evaluated in regards to operability, learnability and user error protection as defined by the ISO 25010 [11].

Pseudo-code has been chosen as the programming language during the challenge. Given the research by Stefik and Siebert [18] emphasizing the barrier to novice students on syntax we thought that pseudo-code would help limit the effect that different experience levels in programming languages could have on the outcome of the data collection. The students that are doing the challenge therefore need no experience in how to program robots in ROS or how HTTP/REST works. There is also a time limit when evaluating the API and the ROS framework. The idea is that using pseudo-code with clear defined rules (and general documentation on the API and ROS framework) will lower the amount of time needed for the evaluation experience.

A. Data Collection Process

The data collection process is set to take 1 hour. During the first 10 minutes the papers were handed out to each person and the rules of the activity were explained. Each student received:

- Pseudocode Conventions
- API or ROS specification
- Session Review

The rules of the activity consisted basically on explaining what the papers they got are. The pseudocode conventions is basic pseudocode rules that can explain different situations that may need to be applied. The specification paper (regardless of API or ROS) contains four tasks to be done. The students are explained that they will be timed but that this time does not

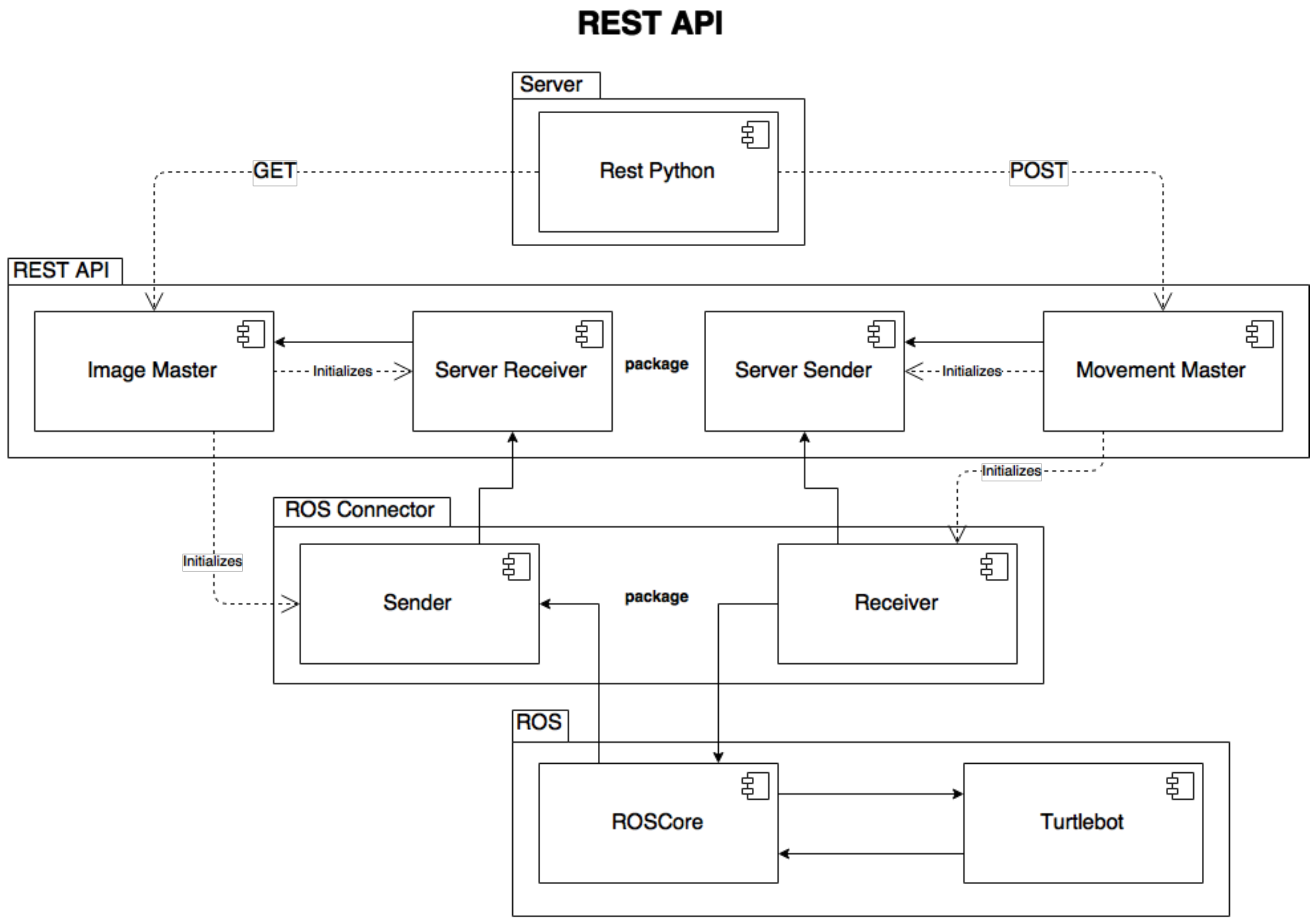


Fig. 2. Component Diagram showing all the packages and components of the API with the connection to ROS and the server.

play any role on their performance. Also each student will notify when they are done with a task and they will receive a time which they will write down in the session review. They are also explained they will be receive a fourth paper when they are done with the first set of tasks. There is a time limit of 30 minutes for this first set of tasks however the students are not aware of this time limit in order to leave this extra stress out of the way so that it will not affect the results. Once the 30 minutes are done or the students are done they receive a fourth paper which is a small program in both ROS and API code. They are explained that both codes do exactly the same thing. They are asked to write down which of the codes they preferred to use and was more readable. After they are done with this last task they can fill out the rest of the session results which contains some questions with some general information. For the code evaluation task they are given five minutes and for filling out the session results paper they are given ten minutes. The last five minutes in the session were used to thank the students and to explain the purpose of the session.

During the session the researchers were allowed to answer questions of a general matter. Students asked some general things about the task and what exactly they were supposed to do but in a general sense there was no specific questions. The most usual answer allowed was to assume anything you wanted and just do the task as stated on the paper to the best of your capabilities. By limiting the response of questions we aimed to limit the bias of the data collected.

B. Evaluation

Evaluation of the study is done in 4 steps. It is tailored to measure the three usability aspects of ISO 25010 [19] that were identified previously: operability, user error protection and learnability. The data collection was designed to elicit data for all variables.

First, questions related to operability were defined. Questions five to eight in the questionnaire had a relation to how operable the system was. It was decided all these questions have a similar weight and therefore each question has a weight of 25 points. The scale on the questions extends from very hard to very easy and is divided into five possibilities measuring the degree of difficulty. Since each questions had a possible scale of 1-5 we chose to give each possibility a 5 point value. Meaning that if the students thought something was very hard they would get five points for that question. If they chose it was very easy they would get 25 points for that question. This score was used to measure operability of the system.

Furthermore errors in the students programming code were identified. The errors to be considered are the misuses of the method calls “Publish/Subscribe” and “Post/Get” for the ROS framework and API respectively, which is specific to the two systems being compared. This is done in order to make a fair comparison of the ROS framework and the API and not the students programming skills in general. However, programming experience can be a risk that was meant to be mitigated by the use of pseudo-code in the tasks. Error identification is to measure user error protection. Table I shows the division of possible methods divided by tool. The errors are based on the misuse of the methods. On the ROS tasks we had a maximum of three possible errors regardless of the task and on the API side we had two possible errors regardless of the task. Since task one had to do with a graphic task then the possible errors are the graphic errors and the general errors. Task two, three and four have only to do with movement and therefore the possible errors are the general error and the movement errors.

The following are the possible commands that were evaluated as possibly wrong for ROS and for the API.

ROS possble functions:

- subscriber = ros.Subscribe(“IMAGE-PATH”, IMAGE-CONTAINER, CALLBACK)
- define callback(image)
- imageVariable.height (Image Data Format)
- object.linear.x (Movement Data Format)
- cmd = ros.Publish(“MOVEMENT-CMD-PATH”, MOVEMENT-CONTAINER, QUEUE-SIZE)
- cmd.publish(TWIST)

API possible functions:

- image = http.get(“IMAGE-URL”)
- width: “VALUE”, height: “VALUE” (Visual Data Format)
- angle:”VALUE”, speed:”VALUE” (Movement Data Format)
- result = http.post(“MOVEMENT-URL”, DATA)

	ROS	API
Graphic Errors	Subscribe Callback Function	Image Get
General Errors	Data Format (Visual or Movement)	Data Format (Visual or Movement)
Movement Errors	Publish instantiation Publish	Movement Post

TABLE I
TABLE WITH METHODS DIVIDED BY TOOL AND BY POSSIBLE ERROR.

The amount of time used to complete the tasks was also evaluated for both ROS and the API. All students were timed during the experience. The time registered is used to measure learnability. It is important to note that only time and not correctness is taken into account. The students were not aware about the significance of time and were told repetitively that it was not a race. They were also not aware of the time limit of 30 minutes and as such we do not expect any race cases in which a student wanted to beat time.

To mitigate risks of biases, both researchers evaluated all papers. Both researchers followed the same rules and compared the data at the end as well as go through the results after the raw data has been coded. When the researchers did not agree on something a middle ground would be attempted. If the numbers did not allow for middle ground both researchers presented their case and a common decision was taken.

V. RESULTS

In this section the results are discussed in terms of the three usability aspects (operability, learnability and error protection) with regards to the API and ROS. The data was gathered during our data collection sessions with students and interpreted with the use of RStudio. There was data correlating to each of the three usability aspects and they all yielded varying results.

When checking normality of the samples with regards to the tasks it was noted that the ROS sample is not normal and the API sample for the same tasks is normal. A nonparametric test was chosen given the size of the sample being so small and part of the sample not being normalized. It was considered that there is only one independent variable with two levels. The nature of the dependent variables is ordinal since it is a numeric value. Therefore according to [20] the most suited test for our purposes is the Wilcoxon-Mann Whitney test.

The results show that there are differences between using the API and ROS when it comes to usability, however the differences were more subtle that we initially thought. The tasks were made to evaluate learnability by the time it took the students to finish each task. When looking at the means of the four tasks that the students had to finish, the mean is higher on average with the students that used ROS, as shown in 8. The variance of the same average was a great deal higher for ROS then for the API. The standard deviation was also higher for ROS in comparison to the API, shown in 4, which indicates that there was more uncertainty in the students doing the ROS challenges. It was initially thought that the ROS tasks would be harder than the API tasks, this could also indicates that they were. When using the Wilcoxx-Mann test on this sample a difference was noted, but not a statistically significant difference in the time it took the students to finish the tasks with ROS and the API. This means that it cannot be said that there is a difference in learnability between the API and ROS. However there was an interesting trend when going through the results. None of the students that did the challenge for ROS finished the last task in the given amount of time.

To measure error protection the number of errors students made on the tasks were counted. The mean was higher for the students that used ROS in comparison to the API, as shown in 9. The variance was slightly higher for the students using the API and the standard deviation was higher for the students using ROS as shown in 3. This is an indication that it was easier to make a mistake with ROS and that within the group of students that used the API there were many people getting

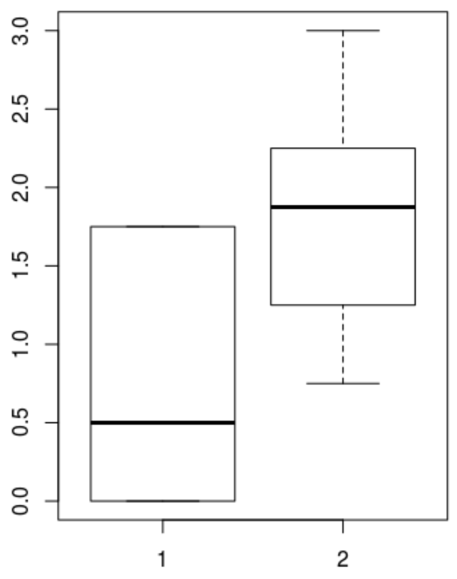


Fig. 3. Shows the average amount of errors for 1. API and 2. ROS

few errors and some that had many more than average. When using the Wilcoxx-Mann test on this sample a statistically significant difference on error protection was found, as shown in 6. This means that the sample was tolerant enough for the tests to be successful and the data from each student showed the anticipated difference.

For the measurement of operability, what the students thought about the challenge as a whole was examined. Questions about key events in the experience were devised where students could express their opinions about the tasks and events surrounding the tasks. It was found that students that were using the API found the tasks generally easier than students using ROS, shown in 5. The students were given two functionality wise similar pieces of code, one with ROS and one with the API. All students that used the API for their tasks found that the API code was easier to understand and would prefer it over its ROS counterpart. However, in students that used ROS for their tasks, six out of ten said that they thought that the API was easier and would prefer it over ROS, shown in 7. This, again, indicates that the API was more readable and easier to understand.

VI. Discussion

Through the results, a correlation was identified between number of finished tasks, amount of errors and amount of time to finish a task. Students that used ROS and finished the tasks below average time, had more errors than the students using the API. This suggests that what the students learned about ROS was not correct or was harder to implement. It could

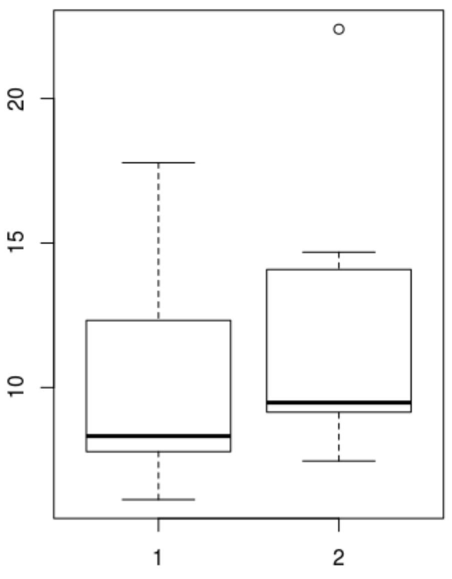


Fig. 4. Shows the average time for tasks done with 1. API and 2. ROS

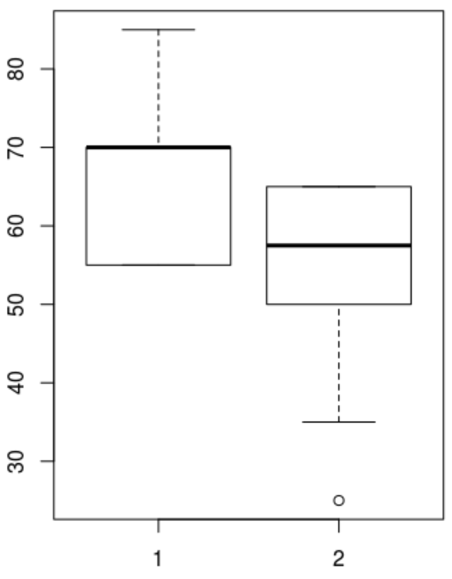


Fig. 5. Shows the average question score for 1. API and 2. ROS

Results	P-Value	Avg. Time	0,4875
		Avg. Error	0,01519
		Avg. Question	0,03762

Fig. 6. Shows the p-values for time per task, error per task and question scores

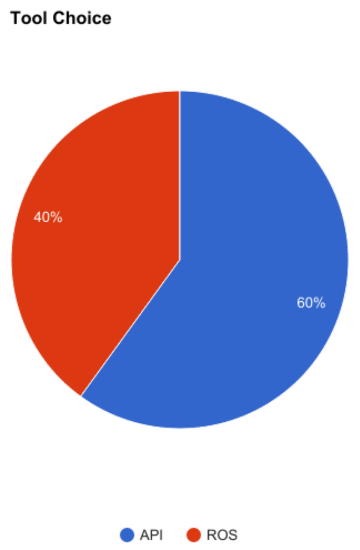


Fig. 7. Shows the distribution of students that chose API or ROS as the easiest tool to use

	Avg	
	API	ROS
Mean	10,18	11,59
Variance	15,94	20,72
Standard Deviation	3,99	4,55

Fig. 8. Shows the mean, variance and standard deviation for learnability (time per tasks)

	API	ROS
Mean	0,81	1,78
Variance	0,61	0,45
Standard Deviation	0,78	0,67

Fig. 9. Shows the mean, variance and standard deviation for error protection (errors per tasks)

also mean that the students that used ROS had a hard time properly understanding the instructions. There was a tendency that students using the API had varying levels of experience while many students using ROS were on a similar level, which could skew the results in some direction. The sample was conveniently selected with students of the same university therefore it was expected to see similar levels of experience. Students that marked themselves as "Intermediate" and used the API had a lower average time to finish tasks than the same group in ROS users, as shown in 10. That could indicate that intermediates find the API easier or faster to learn.

A. API Evaluation

The initial set of rules for declaring the experiment successful was to have at least 2/3 of the usability aspects confirmed. In this case that is true. Both the measurement of operability and error protection gave the anticipated results, however learnability did not. From this it can be concluded that the setting, with students of university of Gothenburg, the API is easier to use than the ROS framework. It is harder to make mistakes with the API in our setting. It is not statistically significant, but the numbers for learnability also suggest that it is easier to learn.

To increase the accuracy of learnability it would be interesting to redo the experiment with some lessons learned about handling groups. During the data collection people were there as volunteers which makes it sensitive when the activity might not be of their interest. Perhaps data collection with individual sessions would have been easier to handle than having the group session.

During our literature review similar data collection processes were not found which makes it impossible to compare with already existing work. This could be due to the general consensus that it is obvious that a simplified API would be more usable than the framework itself. Another possibility is that robotics is a relatively young field which makes a lot of specific topics unexplored.

B. Validity Threats

Threats to internal validity, to construct validity and to conclusion validity have been found. The selection of the sample is one of the threats. Since the sample is homogenous the results may be affected. For the scope of this paper it was not possible to get a random sample which is why a convenient sample was chosen. Having a convenient sample is a threat in itself. Some relationships have been found between level of experience and amount of time taken to finish tasks. This could be a sample issue and it is not possible to verify without a larger sample. Threats that relate to how the data collection was developed and administered are also a factor. Lack of experience from the researchers might have caused the study to miss more qualitative questions which could lead to understand the differences in results of some participants. The documents provided, which include how the ROS framework works, were written by the researchers. The initial thought was to provide documentation from ROS. However, given that the framework

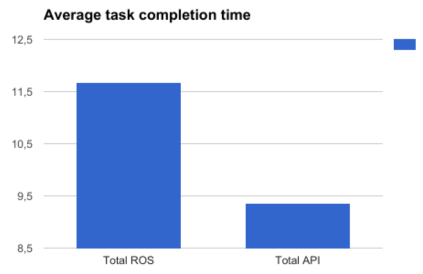


Fig. 10. Shows the average time of ROS vs API of students with experience Intermediate and above)

is documented mostly by examples, it made it difficult to include because it would reveal too much to the participants. The fact that the researchers wrote the ROS documentation could have led the results in a specific direction. Another problem during the data collection was that participants did not arrive all at the same time. The researchers chose to start the activity at the proposed time. Given that more participants were needed the new dropins were accepted and therefore instructions had to be repeated and thus might have an impact on how instructions were given to different groups. Finally, pseudo-code was used to avoid having language specific bias on how the tools worked. Yet pseudocode in itself can be considered a specific language. This might have led to some participants having more experience with the proposed language than others. The main threat however is of course the bias the researchers could have to when evaluating and interpreting the data. Even though rules were implemented to try to avoid such bias it is clear that both researches have a bias which could lead the results to point towards the API being easier to use. Although evaluation is thought to have been objective, it would be interesting to have other researches try to reproduce this data collection with ROS or with any other framework for that matter.

VII. Conclusions

In this paper the process of designing and building an API has been explained. The aim has been set as to abstract the robotics framework away from the user. The API was evaluated based on three usability aspects which are operability, learnability and user error protection. Background to the topics were introduced and what the contributions would be to the current research. The data collection was explained in full detail and could be reproduced with the available instructions. The literature review showed that usability has not been an evaluation method for previous work and therefore this could be considered as new research. The results show that the API can be considered more usable as a single point of contact than contacting the framework directly. The results also show that for more experienced students the difference becomes even more obvious with the API being particularly more usable.

In future work it would be interesting to be able to use actual coding with students in a similar level of experience. This could lead to more accurate values since the code could be evaluated with more accuracy in terms of what is an error and what is not. Furthermore it would be interesting to try to do a communications module with Opendavinci and try to run the same experiment with it and evaluate if the result is similar. This could lead to more robust results given that it is not specific to ROS anymore. It would be interesting to try to apply this type of data collection with several frameworks and generalize the results to robotic frameworks in general. Finally, a more random sample that includes professionals in the area would lead to more robust results and could increase the generalizability of the results.

VIII. Acknowledgment

We would like to thank Piergiuseppe Mallozzi for his help as our supervisor. He guided the way so that this project was a reality. We would also like to thank everyone at the department for their input and their time. Finally our deepest gratitude to the nineteen students who were willing to help us out so that our study could be realized.

References

[1] H. Kopetz, *Internet of Things*. Boston, MA: Springer US, 2011, pp. 307–323. [Online]. Available: http://dx.doi.org/10.1007/978-1-4419-8237-7_13

[2] Y. Chen, Z. Du, and M. García-Acosta, "Robot as a service in cloud computing," in *2010 Fifth IEEE International Symposium on Service Oriented System Engineering*, June 2010, pp. 151–158.

[3] R. Doriya, P. Chakraborty, and G. C. Nandi, "Robotic services in cloud computing paradigm," in *2012 International Symposium on Cloud and Services Computing*, Dec 2012, pp. 80–83.

[4] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, no. 2, pp. 14:1–14:42, May 2009. [Online]. Available: http://doi.acm.org/10.1145/1516533.1516538

[5] B. H. Cheng, H. Giese, P. Inverardi, J. Magee, R. de Lemos, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. D. M. Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle, "08031 – software engineering for self-adaptive systems: A research road map," in *Software Engineering for Self-Adaptive Systems*, ser. Dagstuhl Seminar Proceedings, B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Eds., no. 08031. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2008. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2008/1500

[6] R. de Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, D. Weyns, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cukic, R. Desmarais, S. Dustdar, G. Engels, K. Geihs, K. M. Göschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, A. Lopes, J. Magee, S. Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezzè, C. Prehofer, W. Schäfer, R. Schlichting, D. B. Smith, J. P. Sousa, L. Tahvildari, K. Wong, and J. Wuttke, *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–32. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35813-5_1

[7] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan 2003.

[8] J. Kramer and M. Scheutz, "Development environments for autonomous mobile robots: A survey," *Autonomous Robots*, vol. 22, no. 2, pp. 101–132, 2007. [Online]. Available: http://dx.doi.org/10.1007/s10514-006-9013-8

[9] B. K. Kim, M. Miyazaki, K. Ohba, S. Hirai, and K. Tanie, “Web services based robot control platform for ubiquitous functions,” in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, April 2005, pp. 691–696.

[10] I. A. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I.-H. Shu, and D. Apfelbaum, “Claraty: Challenges and steps toward reusable robotic software,” *International Journal of Advanced Robotic Systems*, vol. 3, no. 1, p. 5, 2006. [Online]. Available: http://dx.doi.org/10.5772/5766

[11] ISO, “Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models,” International Organization for Standardization, Geneva, Switzerland, ISO 25010:2011, 2011.

[12] S. Osentoski, G. Jay, C. Crick, B. Pitzer, C. DuHadway, and O. C. Jenkins, “Robots as web services: Reproducible experimentation and application development using rosjs,” in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 6078–6083.

[13] A. Koubaa, *A Service-Oriented Architecture for Virtualizing Robots in Robot-as-a-Service Clouds*. Cham: Springer International Publishing, 2014, pp. 196–208. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-04891-8_17

[14] ——, “Ros as a service: Web services for robot operating system,” *Journal of Software Engineering for Robotics*, vol. 6, no. 1, pp. 1–14, Dec 2015.

[15] C. Berger, *An Open Continuous Deployment Infrastructure for a Self-driving Vehicle Ecosystem*. Cham: Springer International Publishing, 2016, pp. 177–183. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-39225-7_14

[16] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: an open-source robot operating system,” in *ICRA Workshop on Open Source Software*, 2009.

[17] R. T. Vaughan, B. P. Gerkey, and A. Howard, “On device abstractions for portable, reusable robot code,” in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, vol. 3, Oct 2003, pp. 2421–2427 vol.3.

[18] A. Stefik and S. Siebert, “An empirical investigation into programming language syntax,” *Trans. Comput. Educ.*, vol. 13, no. 4, pp. 19:1–19:40, Nov. 2013. [Online]. Available: http://doi.acm.org/10.1145/2534973

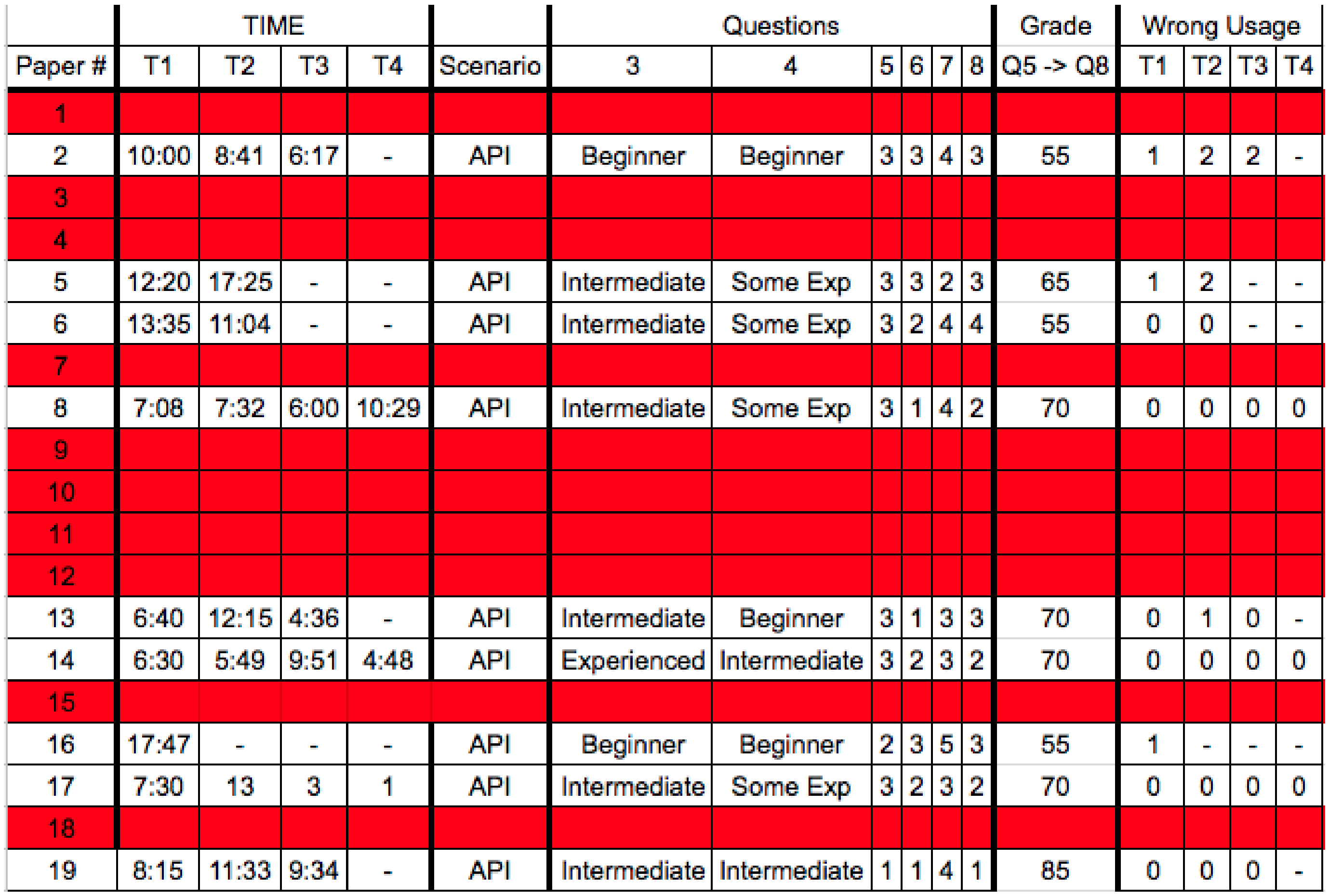
[19] A. Abran, A. Khelifi, W. Suryn, and A. Seffah, “Usability meanings and interpretations in iso standards,” *Software Quality Journal*, vol. 11, no. 4, pp. 325–338, 2003. [Online]. Available: http://dx.doi.org/10.1023/A:1025869312943

[20] UCLA, “CHOOSING THE CORRECT STATISTICAL TEST IN SAS, STATA, SPSS AND R,” http://stats.idre.ucla.edu/other/mult-pkg/whatstat/, accessed: 2017-03-30.

[21] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

Appendix A
Data Collection

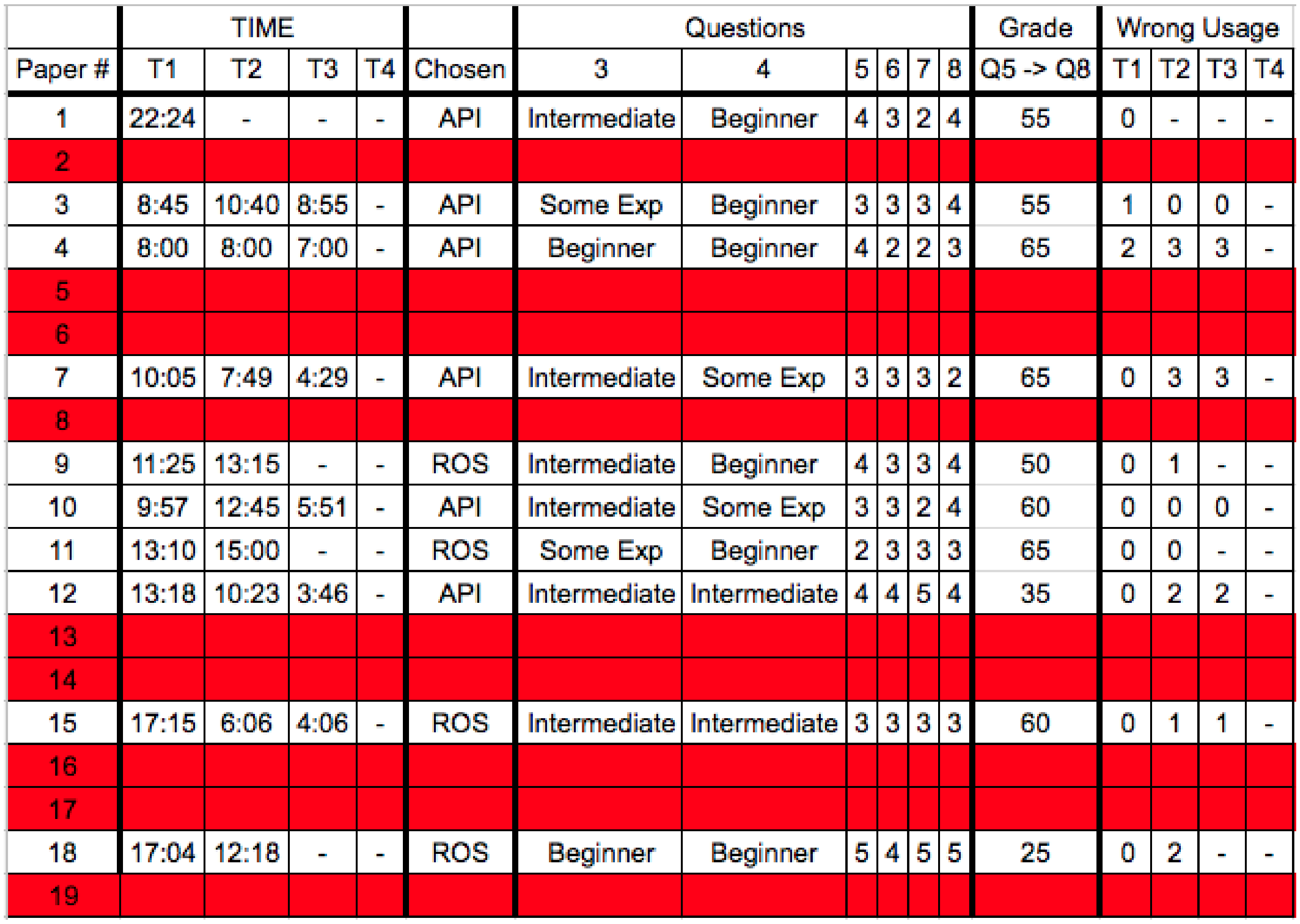
A. Data for the API users



	TIME					Questions						Grade	Wrong Usage			
Paper #	T1	T2	T3	T4	Scenario	3	4	5	6	7	8	Q5 -> Q8	T1	T2	T3	T4
1																
2	10:00	8:41	6:17	-	API	Beginner	Beginner	3	3	4	3	55	1	2	2	-
3																
4																
5	12:20	17:25	-	-	API	Intermediate	Some Exp	3	3	2	3	65	1	2	-	-
6	13:35	11:04	-	-	API	Intermediate	Some Exp	3	2	4	4	55	0	0	-	-
7																
8	7:08	7:32	6:00	10:29	API	Intermediate	Some Exp	3	1	4	2	70	0	0	0	0
9																
10																
11																
12																
13	6:40	12:15	4:36	-	API	Intermediate	Beginner	3	1	3	3	70	0	1	0	-
14	6:30	5:49	9:51	4:48	API	Experienced	Intermediate	3	2	3	2	70	0	0	0	0
15																
16	17:47	-	-	-	API	Beginner	Beginner	2	3	5	3	55	1	-	-	-
17	7:30	13	3	1	API	Intermediate	Some Exp	3	2	3	2	70	0	0	0	0
18																
19	8:15	11:33	9:34	-	API	Intermediate	Intermediate	1	1	4	1	85	0	0	0	-

Fig. 11. Shows the data for the students who used the API.

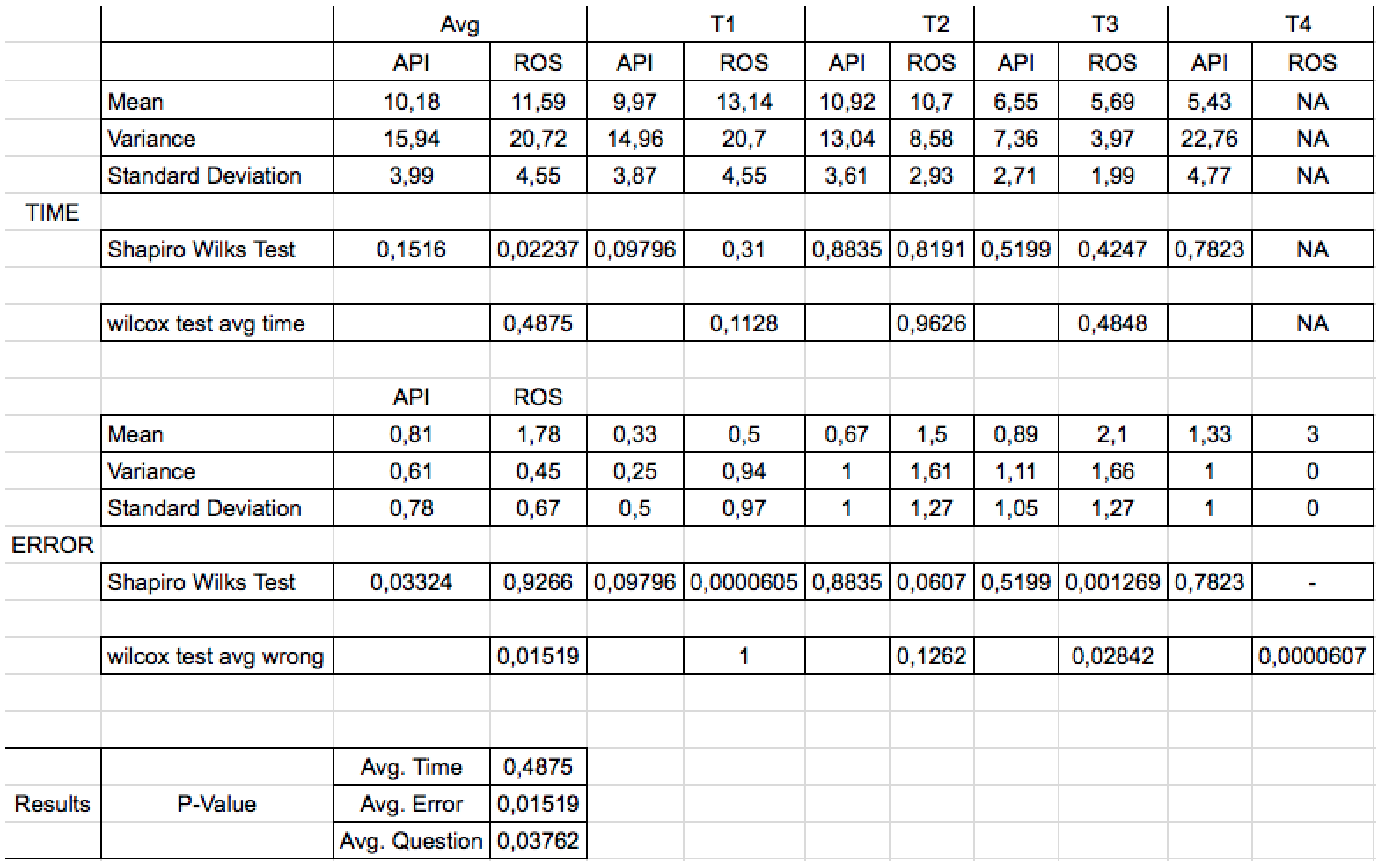
B. Data for the ROS users



	TIME					Questions						Grade	Wrong Usage			
Paper #	T1	T2	T3	T4	Chosen	3	4	5	6	7	8	Q5 -> Q8	T1	T2	T3	T4
1	22:24	-	-	-	API	Intermediate	Beginner	4	3	2	4	55	0	-	-	-
2																
3	8:45	10:40	8:55	-	API	Some Exp	Beginner	3	3	3	4	55	1	0	0	-
4	8:00	8:00	7:00	-	API	Beginner	Beginner	4	2	2	3	65	2	3	3	-
5																
6																
7	10:05	7:49	4:29	-	API	Intermediate	Some Exp	3	3	3	2	65	0	3	3	-
8																
9	11:25	13:15	-	-	ROS	Intermediate	Beginner	4	3	3	4	50	0	1	-	-
10	9:57	12:45	5:51	-	API	Intermediate	Some Exp	3	3	2	4	60	0	0	0	-
11	13:10	15:00	-	-	ROS	Some Exp	Beginner	2	3	3	3	65	0	0	-	-
12	13:18	10:23	3:46	-	API	Intermediate	Intermediate	4	4	5	4	35	0	2	2	-
13																
14																
15	17:15	6:06	4:06	-	ROS	Intermediate	Intermediate	3	3	3	3	60	0	1	1	-
16																
17																
18	17:04	12:18	-	-	ROS	Beginner	Beginner	5	4	5	5	25	0	2	-	-
19																

Fig. 12. Shows the data for the students who used ROS.

C. Results from the data through R



		Avg		T1		T2		T3		T4	
		API	ROS	API	ROS	API	ROS	API	ROS	API	ROS
	Mean	10,18	11,59	9,97	13,14	10,92	10,7	6,55	5,69	5,43	NA
	Variance	15,94	20,72	14,96	20,7	13,04	8,58	7,36	3,97	22,76	NA
	Standard Deviation	3,99	4,55	3,87	4,55	3,61	2,93	2,71	1,99	4,77	NA
TIME											
	Shapiro Wilks Test	0,1516	0,02237	0,09796	0,31	0,8835	0,8191	0,5199	0,4247	0,7823	NA
	wilcox test avg time		0,4875		0,1128		0,9626		0,4848		NA
		API	ROS								
	Mean	0,81	1,78	0,33	0,5	0,67	1,5	0,89	2,1	1,33	3
	Variance	0,61	0,45	0,25	0,94	1	1,61	1,11	1,66	1	0
	Standard Deviation	0,78	0,67	0,5	0,97	1	1,27	1,05	1,27	1	0
ERROR											
	Shapiro Wilks Test	0,03324	0,9266	0,09796	0,0000605	0,8835	0,0607	0,5199	0,001269	0,7823	-
	wilcox test avg wrong		0,01519		1		0,1262		0,02842		0,0000607
		Avg. Time	0,4875								
Results	P-Value	Avg. Error	0,01519								
		Avg. Question	0,03762								

Fig. 13. Shows the results of the data evaluated with the program R.

APPENDIX B
PSEUDOCODE CONVENTIONS [21]

- Indentation indicates block structure. For example, the body of the for loop that begins on line 1 consists of lines 2–8, and the body of the while loop that begins on line 5 contains lines 6–7 but not line 8. Our indentation style applies to if-else statements as well. Using indentation instead of conventional indicators of block structure, such as begin and end statements, greatly reduces clutter while preserving, or even enhancing, clarity.

- The looping constructs while, for, and repeat-until and the if-else conditional construct have interpretations similar to those in C, C++, Java, Python, and Pascal.4 In this book, the loop counter retains its value after exiting the loop, unlike some situations that arise in C++, Java, and Pascal. Thus, immediately after a for loop, the loop counter's value is the value that first exceeded the for loop bound. We used this property in our correctness argument for insertion sort. The for loop header in line 1 is for j = 2 to A.length, and so when this loop terminates, j = A.length + 1 (or, equivalently, j = n + 1, since n = A.length). We use the keyword to when a for loop increments its loop counter in each iteration, and we use the keyword downto when a for loop decrements its loop counter. When the loop counter changes by an amount greater than 1, the amount of change follows the optional keyword by.

- The symbol "//" indicates that the remainder of the line is a comment.

- A multiple assignment of the form i = j = e assigns to both variables i and j the value of expression e; it should be treated as equivalent to the assignment j = e followed by the assignment i = j .

- Variables (such as i , j, and key) are local to the given procedure. We shall not use global variables without explicit indication.

- We access array elements by specifying the array name followed by the index in square brackets. For example, A[i] indicates the i th element of the array A. The notation ".." is used to indicate a range of values within an array. Thus, A[1 ... j] indicates the subarray of A consisting of the j elements A[1], A[2], ..., A[j].

- We typically organize compound data into objects, which are composed of attributes. We access a particular attribute using the syntax found in many object-oriented programming languages: the object name, followed by a dot, followed by the attribute name. For example, we treat an array as an object with the attribute length indicating how many elements it contains. To specify the number of elements in an array A, we write A.length.

- We treat a variable representing an array or object as a pointer to the data representing the array or object. For all attributes f of an object x, setting y = x causes y.f to equal x.f . Moreover, if we now set x.f = 3, then afterward not only does x.f equal 3, but y.f equals 3 as well. In other words, x and y point to the same object after the assignment y = x.

- Our attribute notation can "cascade." For example, suppose that the attribute f is itself a pointer to some type of object that has an attribute g. Then the notation x.f.g is implicitly parenthesized as (x.f).g. In other words, if we had assigned y = x.f , then x.f.g is the same as y.g.

- Sometimes, a pointer will refer to no object at all. In this case, we give it the special value NIL.

- We pass parameters to a procedure by value: the called procedure receives its own copy of the parameters, and if it assigns a value to a parameter, the change is not seen by the calling procedure. When objects are passed, the pointer to the data representing the object is copied, but the object's attributes are not. For example, if x is a parameter of a called procedure, the assignment x = y within the called procedure is not visible to the calling procedure. The assignment x.f = 3, however, is visible. Similarly, arrays are passed by pointer, so that a pointer to the array is passed, rather than the entire array, and changes to individual array elements are visible to the calling procedure.

- A return statement immediately transfers control back to the point of call in the calling procedure. Most return statements also take a value to pass back to the caller. Our pseudocode differs from many programming languages in that we allow multiple values to be returned in a single return statement.

- The boolean operators "and" and "or" are short circuiting. That is, when we evaluate the expression "x and y" we first evaluate x. If x evaluates to FALSE, then the entire expression cannot evaluate to TRUE, and so we do not evaluate y. If, on the other hand, x evaluates to TRUE, we must evaluate y to determine the value of the entire expression. Similarly, in the expression “x or y” we evaluate the expression y only if x evaluates to FALSE. Short-circuiting operators allow us to write boolean expressions such as “x != NIL and x.f = y” without worrying about what happens when we try to evaluate x.f when x is NIL.

- The keyword error indicates that an error occurred because conditions were wrong for the procedure to have been called. The calling procedure is responsible for handling the error, and so we do not specify what action to take.

Sample of insertion sort:

```
INSERTION-SORT($A$)
for $j = 2$ to $A.length$
    $key = A[j]$
    // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.
    $i = j - 1$
    while $i > 0$ and $A[i] > key$
        $A[i+1] = A[i]$
        $i = i - 1$
    $A[i+1] = key$
```

Appendix C
ROS

ROS FRAMEWORK PSEUDOCODE SCENARIOS:

The following are scenarios to be implemented in pseudocode. Read the ROS pseudocode rules in order to know how to implement ROS functionality. Also refer to the pseudocode conventions when in doubt on how to write according to our standards.

1) Get an image from the camera and visualize it on the screen.
2) Have the robot make a square by making turns to the left and forward movements.
3) Have the robot make a rectangle by making turns to the right and forward movements.
4) Have the robot move in the shape of an 8 (moves can be done either by turns left or right or movements forward and back).

ROS PSEUDOCODE RULES:

1) To import the library (get the object "ros"): import ros
2) Make an image **subscriber object: subscriber = ros.Subscribe("IMAGE-PATH", IMAGE-CONTAINER, CALLBACK)**.
   - Returns a subscriber object.
   - Image-PATH is a ROS topic. For the purpose of this test it can remain unchanged.
   - Image-Container is datatype of the image to be received. In this case the format is **Image**.
   - Callback is a function that you have to define which gets called each time you subscribe to an image. Whatever is in the callback function will be executed at subscription time. The callback has as a parameter the variable name for the image.
3) Image Object: The image object is composed of a height, width and data (array containing the image). To be able to use the image object you need to import its library as such: **from sensor_msgs.msg import Image**. To access these variables you called them from the image instance. Ex: **imageVariable.height**.
4) To show an image on the screen use the function **showImg (height, width, array)**. The function requires the height, width and the image array.
5) Make a movement command object (Returns a movement command object): example: **cmd = ros.Publish("MOVEMENT-CMD-PATH", MOVEMENT-CONTAINER, QUEUE-SIZE)**
   - Returns a steering command object which can later be used to call the function publish. Example: **cmd.publish(TWIST)**
   - Movement-Command-PATH is a ROS topic, in this case taking commands for movement. For the purpose of this test it can remain unchanged.
   - Movement-Container is a variable that defines what type of message will be sent. For the sake of this test we will call it TWIST.
   - Queue-Size takes a size of the messaging queue. For example, queue_size=5
6) Making an instance of type TWIST: **object = new Twist()** Twist objects have two variables of type Vector3. They can be accessed as **object.linear** or **object.angular**. Take into account the variable **object** is from the example of making a TWIST instance.
7) Vector3 datatype: The Vector3 object is represented by three points on a 3d space. This object has three variables: x, y, z. They represent a vector in free space. Access the variables by using **v3var.x** or **v3var.y** or **v3var.z**. **NOTE**: when turning positive values are right and negative left. Turning values are in degrees. When moving positive values are forward and negative values reverse. Zero in both moving and turning mean "no action".

Tip: Be careful which value you change (x,y,z) when moving and which value to change when turning. You can also use the "chain" convention by doing **object.linear.x**.

Appendix D
API

API PSEUDOCODE SCENARIOS:

The following are scenarios to be implemented in pseudocode. Read the API pseudocode rules in order to know how to implement API functionality. Also refer to the pseudocode conventions when in doubt on how to write according to our standards.

1) Get an image from the camera and visualize it on the screen.
2) Have the robot make a square by making turns to the left and forward movements.
3) Have the robot make a rectangle by making turns to the right and forward movements.
4) Have the robot move in the shape of an 8 (moves can be done either by turns left or right or movements forward and back).

API PSEUDOCODE RULES:

1) To import a http library for get/post methods: **import http**
2) To import json handler: **import json**
3) To get the values from JSON object just use the common JSON notation. Example:
   myObj = name:"John", age:30, car:null
   x = myObj.name
4) To get an image (Returns a json message with the format: width: "VALUE", height: "VALUE", image: "STRING-ARRAY"): **image = http.get("IMAGE-URL")**.
   - Returns a JSON object with the format above.
   - Image-URL is the URL of the API containing camera images. For the sake of this test it can remain with that name.
5) To show an image on the screen use the function **showImg (height, width, array)**. The function requires the height, width and the image array.
6) To set steering commands: **result = http.post("MOVEMENT-URL", DATA)**.
   - The form of the DATA should be angle:"VALUE", speed:"VALUE".
   - Return the string "success" or "error".
   - "Movement-URL" is the URL of the API that receives all the movement commands. For the sake of this test it can remain with that name.
   - Data is a variable containing movement values, the form of this data is listed above.

   NOTE: when turning positive values are right and negative left. Turning values are in degrees. When moving positive values are forward and negative values reverse. Zero in both moving and turning mean "no action".

Appendix E
API and ROS Scenario

API

```
$importhttp$
$importjson$
$URL = “http://localhost:8080/Camera''$
$imageData = http.get(URL)$
$image[] = imageData.image$
$imageWidth = imageData.width$
$imageHeight = imageData.height$
**if** $(obstacle.searchForObstacle(cv.IplImage(image[], width, height) == True)$
    $angular = “angle : 90, speed : 0''$
    $linear = “angle : 0, speed : 10''$
    $result = http.post(URL, angular)$
    $result2 = http.post(URL, linear)$
    **if** $(result == “success''\&\&result2 == “success'')$
        $print(“Themessagewassuccessfullysent'')$
$displayImage(imageWidth, imageHeight, image[])$
```

ROS

```
$importros$
$importTwist$
IMAGECB$(image)$
    $image[] = imageData.image$
    $imageWidth = imageData.width$
    $imageHeight = imageData.height$
    $cmd = ros.Publish(“cmd\_vel\_mux/input/nav'', Twist, queue\_size = 10)$
    $movement = Twist()$
    $turn = Twist()$
    $if(obstacle.searchForObstacle(cv.IplImage(image[], width, height) == True))$
        $try$
            $movement.angular.x = 90$
            $cmd.publish(movement)$
            $movement.linear.x = 10$
            $cmd.publish(movement)$
        $catch(MessageExceptione)$
            $print(“Error'')$
$ros.init\_node(‘robot')$
$ros.Subscribe(“camera/rgb/image\_raw'', Image, imageCb)$
```

Appendix F
Session Review

Time:
Task 1: ____________ Task 2: ____________ Task 3:____________ Task 4:____________
Scenario: ____________

Questions: Please answer the following questions when you are done with the tasks and the scenario.

1) Please write down which exercise sheet you received: ____________
2) What did you find challenging with your task?

   ____
   ____
   ____

3) Please circle your level of expertise in programming:

Beginner	Some Exp	Intermediate	Experienced	Expert

4) Please circle your level of expertise with robots or robotic frameworks:

Beginner	Some Exp	Intermediate	Experienced	Expert

5) How challenging was it to learn how to use the tools to complete the tasks?

Very easy	Easy	Average	Hard	Very hard

6) How challenging to understand were function and variable names in comparison to what they were meant to do?

Very easy	Easy	Average	Hard	Very hard

7) How challenging did you feel it was to make a coding mistake with the provided tools?

Very easy	Easy	Average	Hard	Very hard

8) Overall how challenging did you find the experience with the tools provided?

Very easy	Easy	Average	Hard	Very hard

Additional Notes:

____
____
____
____