SIMULATION OF THE HUMANOID ROBOT SCOTT

University of Plymouth

Vasilescu Iulius-Alexander ID : 10634586

A report submitted towards the award of BEng degree in Robotics

May 22, 2021

0.1 Abstract

Robot simulations provide great advantages over just mechanical tests, spanning from safety of personnel and equipment, to enabling multiple teams to work on the same robot. There are a variety of simulators, each with their own benefits and drawbacks.

The University of Plymouth has had a history of humanoid robots participating in competitions, from 2004 to 2015. Now, RoboSoc is reviving the project, starting with Scott, which stood for the first time in Feb 2020. Standing at just above 80cm, it features 20 joints and a camera on its head, but lacks hardware, sensors, and has no written software.

A simulator can be used to develop control loops and algorithms. This can be then used to improve the hardware design and the choice of sensors, without access to the robot. This project presents the implementation of the robot into a simulator, connecting it to a controller, as well as a comparison between two simulators, CoppeliaSim and WeBots, focused on their practical viability for undergraduate humanoid robots, and the making of a streamlined approach for the future development cycle of a humanoid robot. Key challenges are finding time efficient ways of improving the development cycles, finding the online resources, implementing the robot in the simulator, connecting the Robot Operating System, as well as writing and compiling relevant information for the project and for the simulator comparison paper.

Keywords: Humanoid Simulator RoboCup ROS CoppeliaSim WeBots.

0.2 Acknowledgements

The author would like to thank the project supervisor professor Mario Gianni, for taking the time to debug, provide great insight and expertise into the software and follow along with the project.

Additionally, the author would like to thank the mentor support provided by co-supervisor Oliver Smith, who had significant relevant feedback at each stage of the project, providing great pointers for the direction of work, as well as remarkable insights into better time management and psychological management of such a project.

The author would also like to acknowledge the support of the WeBots and ROS communities, as well as the authors of the URDF exporting scripts used and referenced in this project. Their work has proven invaluable in terms of time saving and automating development processes.

The author would also like to thank his family members and close friends for their continued support,and for their patience in dealing with me in the current conditions.

0.3 Glossary of acronyms and abbreviations

API - Application Programming Interface

CoppeliaSim - Robot simulation software provided by CoppeliaRobotics

Webots - Robot simulation software provided by Cyberbotics

ROS - Robot Operating System

RViz - ROS Vizualiser

CAD - Computer Aided Design (software)

URDF - Unified Robot Description Format. URDF is an XML file format, used in ROS to describe the robot's elements and properties

XACRO - XML Macro Language. XACRO is a scripting mechanism that allows code modularity and re-use, and is compiled into a URDF.

PROTO - Webots file describing a (robot) node.

TAROS - Towards Autonomous RObotic Systems Conference, held annually

IMU - Inertial Measurement Unit

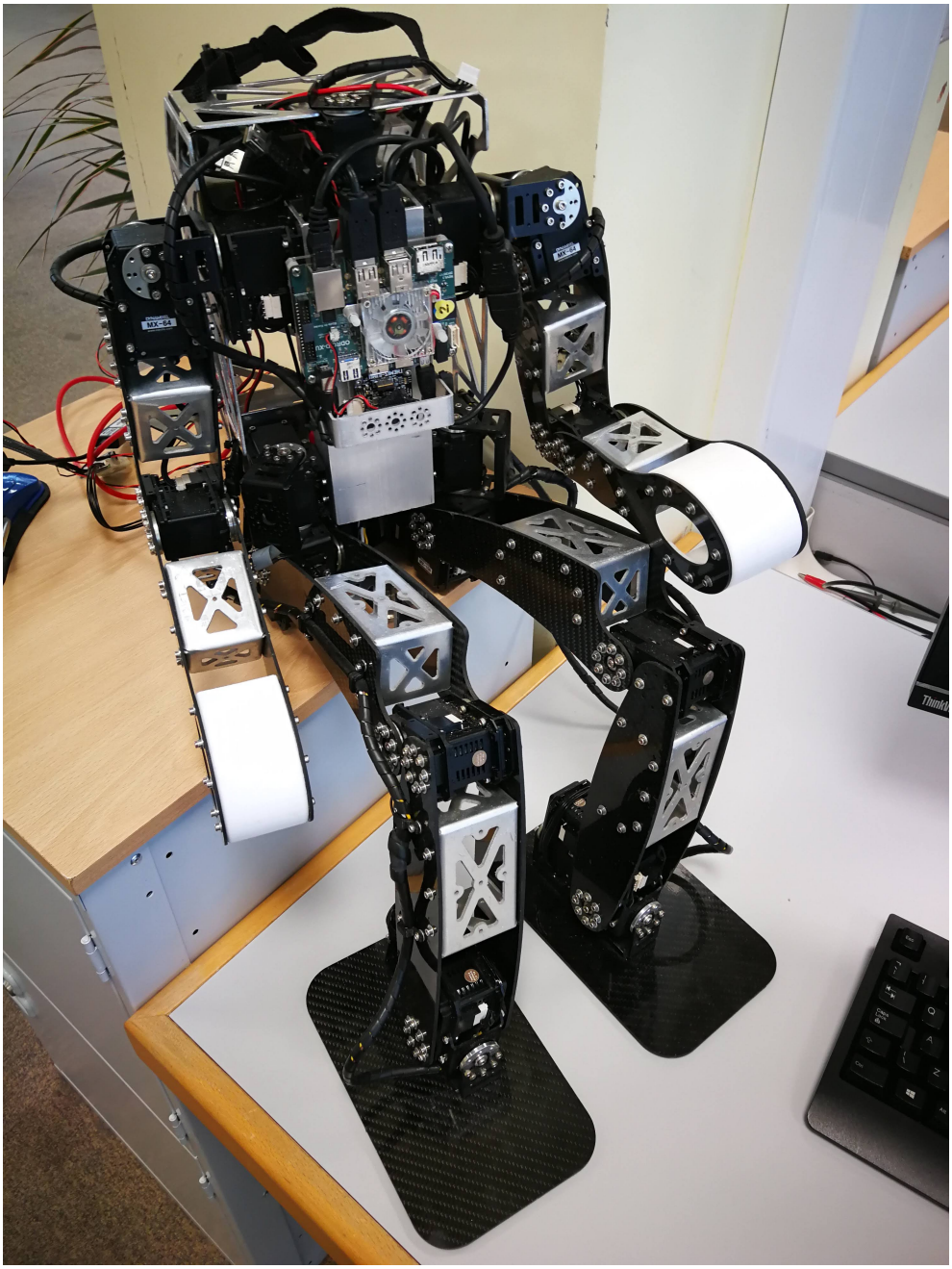

Figure 1: Scott, taking a break

Contents

0.4 Introduction

The project is divided in 3 major purposes, the first of which is to create a simulator for the humanoid robot Scott, which will be used for future development. The second purpose of the project is to detail a comparison between two robot simulation software, CoppeliaSim and WeBots, with focus on their use in undergraduate research. The third purpose of the project is the creation of a simple and efficient development cycle for future development of a humanoid robot. This is done through the creation of a tutorial that significantly lowers the knowledge level needed to make significant contributions to a robot's design, and lowers the knowledge barrier of entry to robotics simulations.

Robot development usually requires a large investment of both time and money, investment that threatened by running tests directly on the hardware. Running the robots without prior tests can also put the researchers at risk. As such, simulators provide a safer and cheaper environment through testing virtual models of the robots and their environments into physics and kinematics simulations. They allow a faster development cycle, while enabling simultaneous work by multiple people from remote locations. This is even more relevant now, put in context of the coronavirus pandemic.

Developing a robot with a simulator requires a 3d model of the robot, a simulation environment and a form of robot control simulation. On the hardware side, the robot might also have a control board, running a separate low-level framework, that addresses only the sensor and motor commands and feedback, in order to eliminate delay caused by the central computing unit. In this project, ROS is used to cover both the robot control simulator, as well as the on-board robot control.

0.4.1 Important software used

ROS

According to the official website[4, 5], it is a framework for designing and testing robot software, that greatly simplifies the task of building software for a robotic application. In practice, the Robot Operating System is a generic robot design tool with a lot of preset packages to help with writing software. The two major advantages is that the system is generic, so it can be adapted for any configuration of a robot, and it is distributable, meaning it can operate from different hardware at the same time. This also means it can be set to run paired with a simulator, and the code it uses is identical with the one you upload on the physical model.

Scott Robot

Scott is a humanoid robot originally developed for RoboCup competitions within the University of Plymouth. The competition presence has seen a hiatus since 2015. In 2019 it has been taken over by the Robotics Society, for further development. At the time, it only had a mechanical body and motors, without sensors or any software. [1, 2, 3]

CoppeliaSim

CoppeliaSim is a robot simulation software originally created by Toshiba R&D and currently developed by a small company from Switzerland called Coppelia Robotics AG. It is designed as a kinematic-dynamics hybrid, and built around a distributed control architecture. [6, 7, 8]

WeBots

WeBots[9] is another robot simulation software, created at the Swiss Federal Institute of Technology and further developed by Cyberbotics Ltd. WeBots includes a large collection of tutorials and assets ready to use, as its architecture is focused more on the physics solid simulation, separating the graphical and physical properties of all assets, as well as providing a more streamline approach to development and setup.

On their homepage, they showcase a number of ongoing collaboration projects, including being the official RoboCup simulator, and partnerships with the likes of Renault, the INTRA group, OPTIMA EU and SNSF. They also show a list of impressive academia and industry companies using their software, like MIT and UCLA, Thales and Samsung.

0.4.2 Simulator Considerations

As stated in "On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward" from 2021 [10] , 3 major opportunities for the use of a simulator are: data generation for machine learning, accelerated engineering design cycle and virtual testing and verification environment. To put them into condition statements, they can be rewritten as:

- ROS compatibility- for data generation and processing, as well as control software

- Simulation must have physics - true testing value
- Simulation must be repeatable - true testing value
- Active community (recent activity) - great support enables fast development
- Software speed and efficiency - efficiency of the simulation, and certain options must be present

Besides these major conditions, in an effort to avoid the issues presented in ”A Study on the Challenges of Using Robotics Simulators for Testing” [11] from 2020. Other relevant points are:

- ease of use
- setup difficulty
- active developer team
- direct model export functionality
- other humanoids as tutorials

NOTE: use a citation/multiple for the consideration list. In this paper, the software used and compared is CoppeliaSim and WeBots.

Other simulation software found to be used in research and industry and taken into consideration is: Gazebo, Unity, RoboDK, OLP Robotics, OpenRAVE, USARsim, ADAMS-MSC, Unreal Engine 4. This are analysed more in-depth in the paper draft attached in the Appendices. The most recent version of it will be linked in the github. Other simulation software can be found in this review covering 2013-2020 papers [12].

0.4.3 Project management

This project has been carried out over the course of approximately 9 months, and the inspiration for the project came from previous work done as part of Robotics Society. Some of the initial modelling phase took place during the second year of the degree, while the majority took place during the final year.

To ensure proper project management, a logbook was kept in OneNote, detailing the research, as well as personal notes, mentor meetings, notable sources of information, project updates and ideas. As a separate set of files, a ”work replication” section has been recorded, to allow replication of the entire project in the event of a major software malfunction. This section also details lists of frequently used commands.

The logbook has proven its usefulness a number of times, with the occasion of reinstalling the entire software suite and remaking all the setups, in addition to serving as an inventory for useful commands and as a record of meeting notes. It was also used for writing the report, the paper, and preparing the tutorials.

0.4.4 Original project milestones

The original project was divided in 4 milestones, and 3 bonus milestones.

- Map the robot into a digital environment
  Create a 3D model of the robot in the preferred environment.
- Import the 3D model into V-Rep and ROS.
  Confirm ROS import using R-Viz.
- Connect basic ROS packages needed for a humanoid robot.
  List ROS Packages here *for setting reading and controlling joint positions.
- Connect V-Rep
  Show ROS connected to V-Rep (input joint controls from ROS, show in V-Rep) and set up V-Rep physics.
- Implement a basic walking gait (**bonus**)
  Create a ROS package using IPM and ZMP feedback control. Implement in V-Rep an IMU and get motor feedback.
- Create a general robot controller in V-Rep (**bonus**)
  Create high level user commands such as ”move forward” or ”turn”, and show them working on the robot.
- Investigate walking gaits (**bonus**)
  Look into more advanced walking gaits, and how they could be implemented into the simulator, if there is time left in the project.

0.4.5 Major milestones achieved

Major milestones achieved have been:

- Initial model measurements and CAD design
- ROS import of the robot
- CoppeliaSim import of the robot (Failed)
- WeBots import of the robot
- WeBots - ROS active bidirectional connection

It is important to note that, despite achieving each milestone at vastly different times, vast amounts of work was still done on each of them afterwards. For example, the CAD design has seen multiple iterations, in order to fit with the exporting software, as well as each of the software that imported it.

0.4.6 Major deviations from initial proposal

The project was initially meant to focus on CoppeliaSim alone and advance the simulation testing further. However due to being unable to connect ROS with CoppeliaSim, for legitimate reasons, in spite of spending a long time on the issue and requesting help from the developer and also receiving a lot of help from the mentor, CoppeliaSim was dropped as the simulator-of-use in this project, after consulting with the supervisor and co-supervisor. This has taken a large portion of the project's time. The decision to change the direction of the project has been strongly considered, and only after both the mentor and supervisor's advice and positive feedback, has been followed through.

As stated in "A Study on the Challenges of Using Robotics Simulators for Testing", [11], the survey shows some clear points that need to be solved in order to efficiently use a simulator in a robot's development. Such points include being able to easily set up a robot and it's environment. This project approaches that exact problem. As quoted in the survey, URDFs are great time consumers, tedious to write by hand and error-prone. Considering that humanoids are notoriously complex, that issue might be one of the biggest a simulation setup has to face. Setting up a connection to ROS, a headless (no render) simulation and setting up the environment are a number of other aspects also approached here.

As such, the focus of the project is to achieve a development cycle that is as generic as possible, from the first design to the last simulation. The focus is not to create just a functioning simulation of a particular humanoid, but to also be able to adapt to future developments. The final goal of this project is to be used by the undergraduates part of the Robotics Society, in order to set Scott up for competitions[23], and for post graduate ongoing research on compliant and bipedal locomotion, undertaken by the mentor at the time of writing.

With that in mind, besides modelling and literature research, most of the work presented here went into exploring and testing the software as in-depth as possible, and making a map of how to set up a simulation with as little manual setup as possible. The process and results of the exploration are detailed in the logbook, which is linked in Appendices.

Due to the number of issues encountered in the project, and the lack of simulator comparisons done from an undergraduate point of use, another secondary aim of the project was added: the creation and submission of a comparison paper to TAROS. The supervisor and co-supervisor have seen value in this research to the wider academic community, and have recommended to pursue a paper application to TAROS - Towards Autonomous Robotic Systems Conference, on the topic of a comparison between simulators, from an undergraduate point of use. A 4 page extended abstract is to be submitted, and a draft copy is available on Github, which will be followed by the final version upon submission.

0.5 Part I: Scott Simulator

0.5.1 Project development as of May 17th 2021

The simulation environment has been fully connected to ROS, enabling easy development of ROS control packages for the robot. This includes having an active motor control and feedback loop, as well as sensor feedback. This is further described and shown in the Software section.

Choice of software

Autodesk Fusion 360 was chosen for the modelling tool because, while Autodesk provides an encompassing suite of industry-level design software, Fusion360 presents the most advantages: cloud based storage, easy sharing, smoother learning curve, machining integration, scripting sockets, as well as the being the author's most well known 3D design software.

Windows 10 and Ubuntu 18.04 Bionic Beaver have been used as operating systems. The first has been used for robot modelling, and the second for use of CoppeliaSim, WeBots and ROS Melodic. All licenses used are either free or educational. Ubuntu 18.04 and ROS Melodic have been chosen under the advice of the supervisor.

Other **important** scripts used are:

"Fusion2Urdf"[13, 14] - by Used to export URDF, XACRO, launch files and meshes from Fusion 360. As they are not related to the Fusion team, the script's authors have been messaged and thanked, and at the time of writing one branch of the project is still ongoing development on GitHub.

"URDF2WeBots"[15] Used to convert xacro and meshes to PROTO format. This is provided by WeBots.

0.5.2 Development cycle

In order to create and set up a robot simulation, using either WeBots or CoppeliaSim, the steps are:

1. create 3d model in a software of choice
2. write a description file (urdf, xacro or proto), containing the meshes, the joints, the hierarchy of the links and the properties and exact position of each link and joint, **with at least 3 decimal precision**. For enabling physics in simulation, mass and inertia center has to also be set up, which uses 6 decimal precision.
3. import the meshes and the URDF files into ROS and check R-Viz by writing launch files
4. import the robot model (meshes and proto/urdf files) into simulator
5. set up simulator environment - enable physics, set up lights, objects, and all the simulation constants
6. connect ROS to simulator and check the connection with manual commands
7. test simulation - repeat the simulation with variations in the settings, to isolate any sources of problems. test for the simulation to be repeatable, for the robot to not have issues with its collision meshes (which typically result in it vibrating or falling on the side), etc.

As an example of the complexity of the URDF/XACRO writing process, for each link 6 triplets need to be precisely defined, and for each joint another 2 triplets, and the joint limits. For the presented robot alone, this totals around 150 triplets with most of then having a precision of 4 decimals or greater, in the case presented here. Any errors can throw off the final physics and collision simulation.

As such, even with a simple humanoid, it becomes apparent why the adoption of simulation software can be slow in the industry and undergraduate research.

Each step of the process has been first completed manually, then researched for ways to automate it or make it more efficient.

3D model

Scott has 18 working joints, 19 links, and over 50 components.

The process of replicating the real world model into the digital version can be exemplified with the process for a single piece of those components: if the piece is flat, a sheet of paper is placed underneath, then the contour is being traced onto it, after which the sheet of paper is placed with as much precision as possible on top of the piece and holes marking the positions of screws and axes are made. Then the paper is placed flat on a work surface, with a measuring tool on top as reference, then photographed. It is important when taking the photos to be as centered as possible to remove the fish-eye effects of the lens. The photos are then imported in Fusion360 in a new file as a canvas, being used as reference. The canvas can be precisely calibrated using the measuring tool's markings.

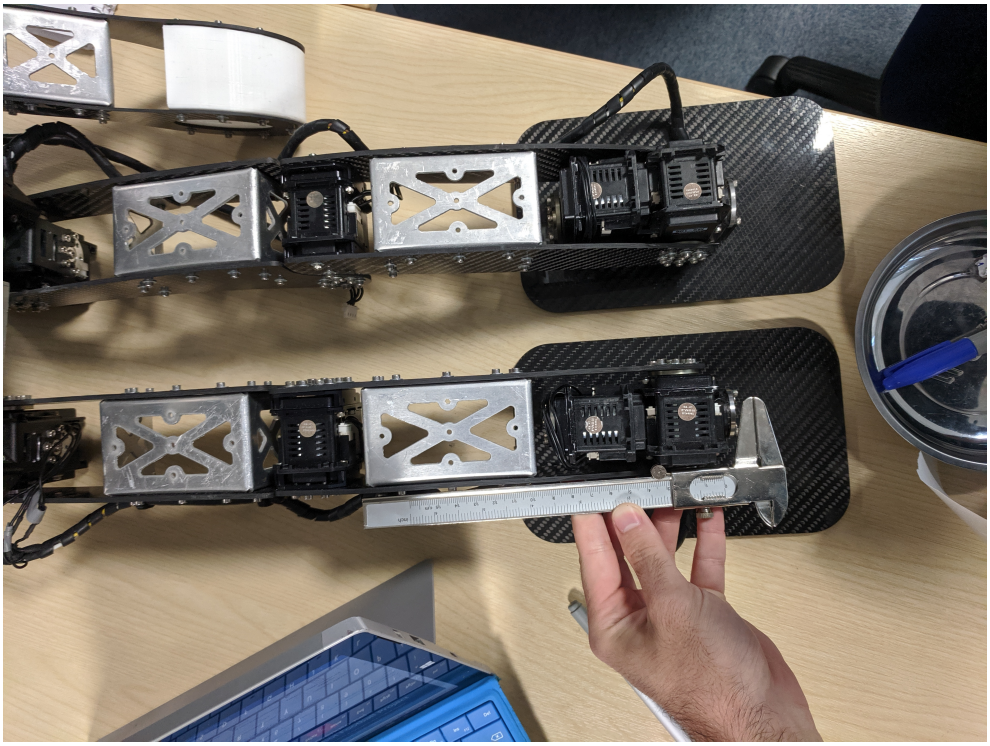

Figure 2: Reference pictures for precision modelling

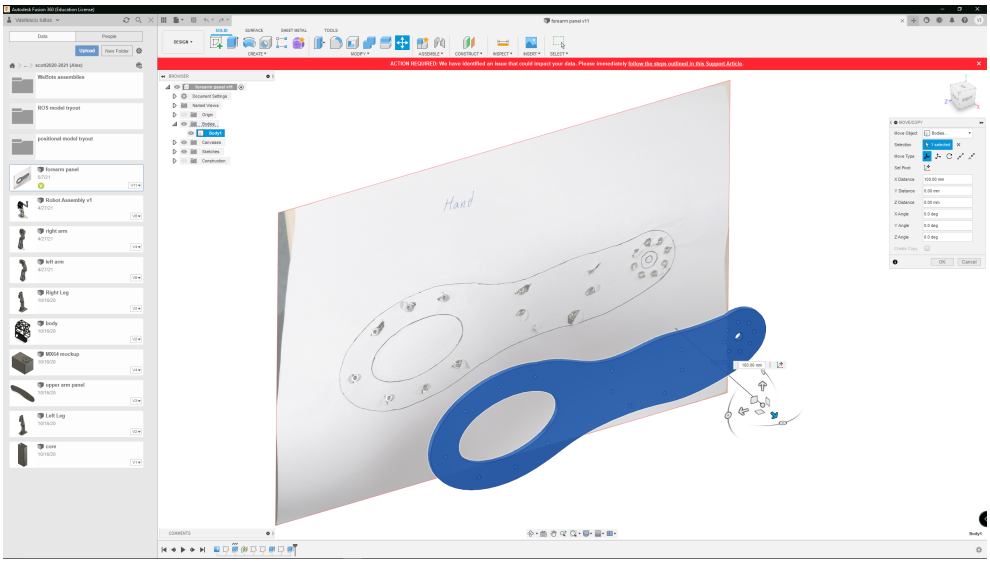

Figure 3: Fusion: forearm mapping from canvas model to solid body

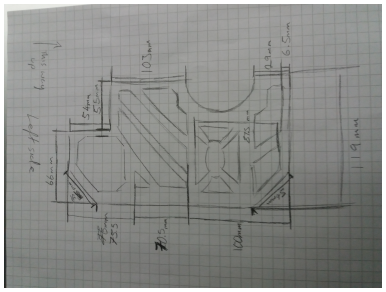

Figure 4: Folded steel bracket - unfolded paper drawing

The robot is mostly made of carbon fiber panels, connected with motors and motor brackets, and folded steel brackets. While all the panels have underwent the described process, the motors have been imported from online libraries, and the brackets have been manually measured and modelled.

Due to the robot's complexity, certain rules for development of late iterations have been designed and followed, with significant time gains. These are documented in a longer format in the logbook. This mostly concern ways to setup a new robot iteration in Fusion360. Example of such a development rule is to first import all the components and recursively remove the links to original parts, before starting setting up joints. Or for manual decimation (the process of combining components into single links, in order to make the simulator physics evaluations significantly faster), it is recommended to use the motors as the main parts, and combine the components so that motor and its actuated component share the same link. To achieve that, you first create the central body piece, which will contain multiple motors, and start a new component each time you pass from a motor to an actuated piece. These rules have saved a lot of time during iterations.

Figure 5: Model, after (the first) decimation. From 50 components, to just 23.

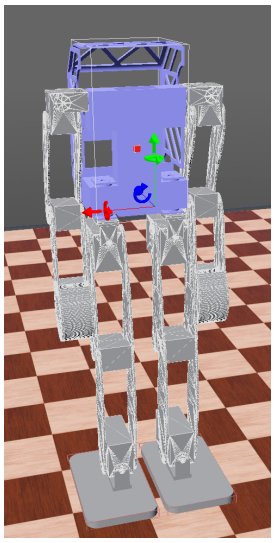

Figure 6: WeBots final meshes. the wireframe represents the collision meshes. Notice the main body, and the feet, are modelled as a cube primitive, and have just the outlines.

Due to the software used for exporting, certain limitations in design and hierarchy have been discovered, recorded in the logbook, and iterations of the design made in order to cover the limitations. One such major limitation is the inability to have joints within the components, while the components can contain other components. Another limitation is the need to have exactly one "base_body", from which the hierarchy of the robot is detected. if the base component has no moving joints, it will be exported alone, ignoring rigid joints. This is solved by combining into a single component all root parts of the kinematic chains. Without resolving all the limitations, the software will throw an error or simply refuse to export anything.

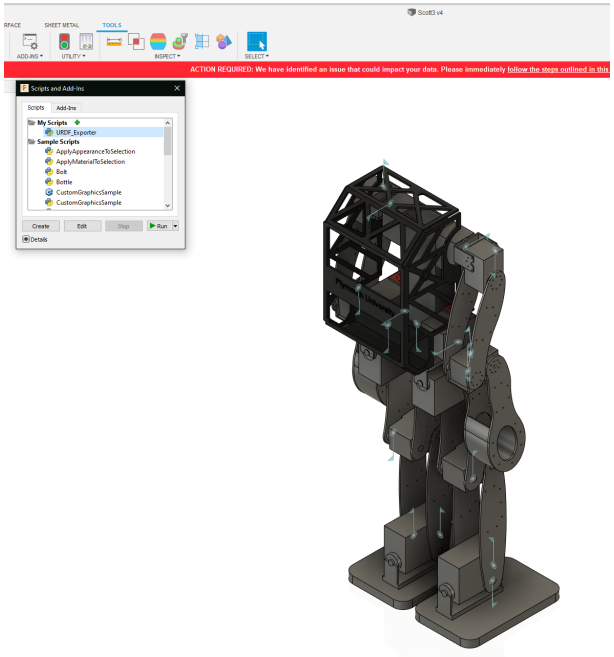

Figure 7: Final WeBots iteration in Fusion, shown with the URDF Exporter script

Robot descriptors: URDF, XACRO, PROTO

A Fusion360 to URDF exporting script was found and used to export all the joints both as 3D meshes and structure files. Both URDF and XACRO have been used in the project to describe the structure of the robot, when importing the meshes.

The same script provided R-Viz and Gazebo launching scripts. It has some limitations, like being unable to import joints from sub components, and ignoring components that are not attached.

At the time of writing, the Fusion-URDF script used cannot be found easily on google without the exact name of the author and project, due to it's very recent development and low use.

Import and settings

Webots' importing process has to go through the URDF-to-PROTO script, provided by their team, which can then be added to the scene from the main window. Creating a new environment and new settings is straight forward, quick, and explained in the tutorials. To later connect to ROS, the robot's controller parameter must be set to "ROS", instead of a custom controller.

In order to showcase the use of sensors, an IMU - inertial measurement unit, used for measuring angular acceleration and inclination - has been placed on Scott's chest. This can be easily done directly within WeBots.

R-Viz can be used to check the files achieved so far (namely the XACRO/URDF), and the Fusion exporting script automatically generates launch files for it.

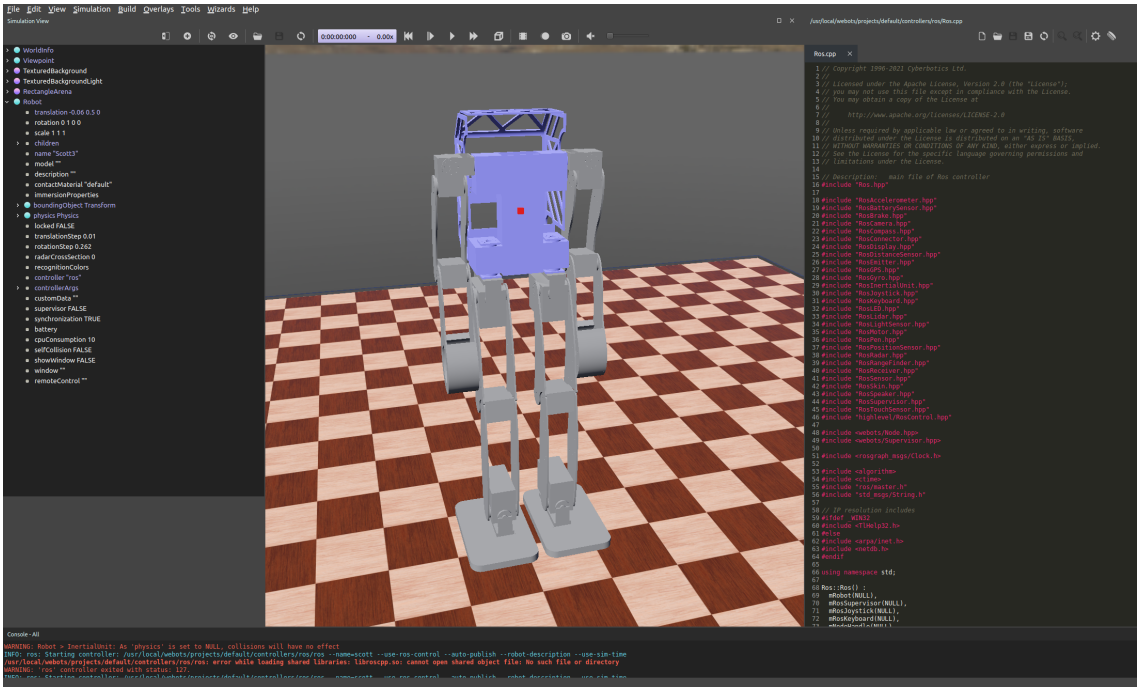

Figure 8: WeBots view. On the left, the ROS controller is selected. On the right, the live editor of the controller. With red, the IMU, placed on the chest.

ROS architecture

A ROS node is a basic executable program running inside the application. It can subscribe and/or publish messages to ROS topics. ROS topics are somewhat like advisor boards, in that nodes can post or read messages from any number of topics, and each topic will only hold messages of a single type. ROS services are a way for nodes to submit specific requests and receive a response. ROS topics are a many-to-many connections, while services are one-to-one connections.

WeBots' standard ROS controller creates a node encapsulating a vast list of services and topics.

The most important services available for a single given motor are the motor sensor enable, which activates the respective topic publish, and the set-position, set-velocity and set-torque, which are the most used for controlling the motor. Other more sensitive controls are available as well, such as setting PID - proportional integrative derivative control - values. A full list is available in the appendix.

The standard ROS controller also creates services for other options, such as robot settings, battery sensors, joystick and keyboard control, and a set of options for each connected sensor. The full list of non-motor services is available in the appendix.

The inertial measurement unit is a sensor added to the robot to test the use of sensors with ROS. Notable in the list is the /robot/get_number_of_devices, as it returns a full list of the active joints and sensors.

To enable a sensor/motor sensor feedback, their "enable" field must be set to true. This is done with:

```
> rosservice call /Assembly2v2_2829_ubuntu/RAnk2_sensor/enable 1
```

After this the controller start publishing the respective value to an identically named topic. To show the stream **rostopic echo ¡name¿** can be used.

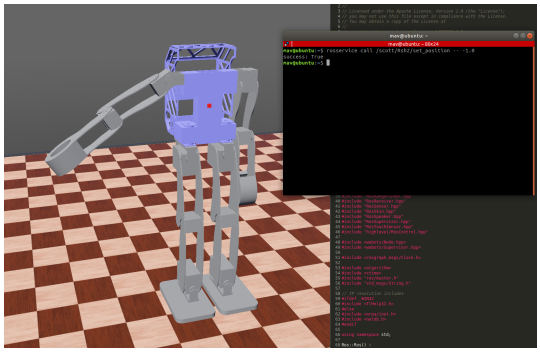

Figure 9: WeBots movement of Rsh2 - Right shoulder joint 2

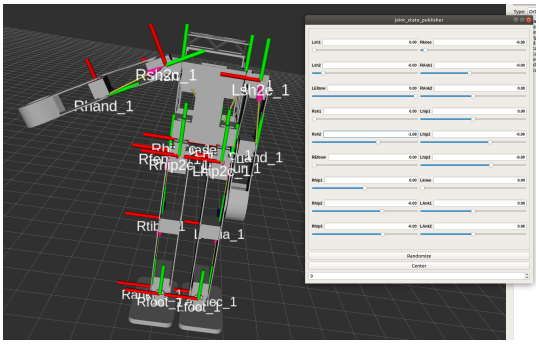

Figure 10: R-Viz (ROS) movement of the same joint

To start a motor movement, there are a number of choices of actuation: move based on it's internal PID controller[24] using the **set_position** command, or a velocity can be specified. Limits such as torque, acceleration, and control values like the PID can also be modified similarly.

To internally represent the state of the robot, ROS has a package called "robot_state_publisher"[16], representing the entire kinematic chain and poses in time frames of the robot. It frequently publishes the current state to tf2[18], a transform library which allows the user to keep track of multiple coordinate frames in time. This is computed based on the provided URDF and another package called "joint_state_publisher"[17]. The latter one contains messages representing all the joint values of the robot.

By default, the WeBots "ROS" controller does not publish its joint values at all. To achieve a unified understanding of the robot's situation, between the simulator and ROS, they need to be connected. This can be achieved in two ways:

A) (recommended) use ros_control, which will automatically enable feedback of all sensors and motors, and the motor commands must go through trajectory_msgs/JointTrajectory topic or control_msgs::FollowJointTrajectoryAction action.

B) without ros_control, using low-level access with the set_position service. Using this method does require to make a plug script to export the joints feedback to the TF package, and to first enable the motors and sensors feedback.

0.5.3 Current issues and the next step

The community is actively developing, and as such certain very specific topics can sometimes be harder to find information on. Publishing this project's findings will cover a couple such topics. An example of such a topic is how to connect ROS and WeBots without writing dedicated non-generic nodes.

WeBots developers provide limited support via their forums and main discord channel, but also provide dedicated high quality support using one of their premium price plans, and professional consulting on demand.

The next step of the work will be to give the documentation to the Robotics Society, where the first development will be the creation of new ROS packages concerning control software. The documentation includes a github, linked in the appendix, which includes manuals, and tutorials which will be posted on Youtube. Its purpose is to make it easy to reproduce and use by undergraduate students, even with minimal experience of ROS and simulators.

0.6 Part II: CoppeliaSim

0.6.1 Development as of May 17th

The successful import of the robot has been achieved, including joint control and correct mesh positioning, and basic physics tests.

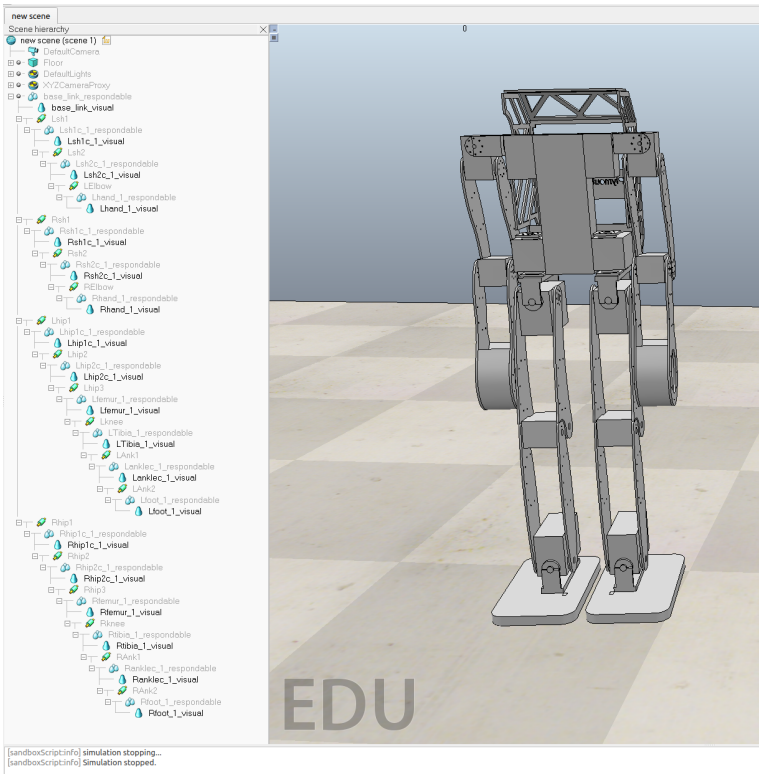

Figure 11: Scott in CoppeliaSim, paused simulation

0.6.2 Current issues

The main issue arose when trying to connect CoppeliaSim with ROS. According to their guide[19], a package called simExtROS[20] needs to be installed and after compilation as a ROS package, some of the resulting files need to be moved to the CoppeliaSim installation folder. Some of the commands presented have simply returned unsolvable errors. The process of solving a great deal of this errors is documented in the logbook. The developer of the project has been unresponsive, and the community help found online has failed to provide any solutions.

After a lot of time spent with this issue, and after consulting with both the supervisor co-supervisor multiple times, the simulator was declared unviable, due to being too complex to correctly set up.

A brief inventory of the issues that have been solved in the process of trying to fix the package:

- cmake update. Ubuntu 18.04 comes with an older package of cmake, that will, by default, refuse to update. to succeed in updating it to at least 3.16, the best course is following the guide from KitWare APT Repository[25].
- ROS melodic needs python 2.7, and to have that version as the system's default. However, for running that package, python 3.8 is required. This resulted in installing several versions, and learning how to select which python version is the primary one, both for python 3 and python 2.
- several packages are both available to install with pip3 (a python3 based installer) and with normal apt commands. however, the two versions are not always the same, despite sharing the name.
- due to requiring to run the package with a number of complex arguments, learning various intricacies of running cmake was required.
- in order to correctly understand, asses and address a given issue, evaluating the dump log of failing to run the package was needed. This included going through hundreds of lines of various dependencies, and figuring out where the error actually happened, instead of where it was last reported.
- the version of libPlugin used - a plugin used by CoppeliaSim -, which has no version tag, and various official forum pages suggest using either the default one, or reinstalling the latest one from the github page.

Major information sources for dealing with the issues have been:

- The supervisor, as they have a vast experience with CoppeliaSim
- official forums of the software
- running through the code from each package and trying to understand where the errors occur.
- the creator of the plugin and package has been emailed, to the date of writing this still without an answer.

An important moment during trying to solve the issues occurred when the author succeeded in purging python 2.7, breaking ros and even some of the system functionality in the process. This led to making great use of the logbook and work replication, as setting up another 2 virtual machines with the same settings was straightforward and fast.

0.7 Part III: WeBots

0.7.1 Development as of May 17th

Two simulation setups have been completed.

The first is an import and basic controller test, that was done in order to investigate as many of the possibilities and workings of the software as possible. It also contains a simple controller module programmed in order to demonstrate some of those features, like real time simulation, faster than real time simulation, consistency, GPU use, and headless simulation (with no rendering). These points can be seen in the shared YouTube video presentation**??**.

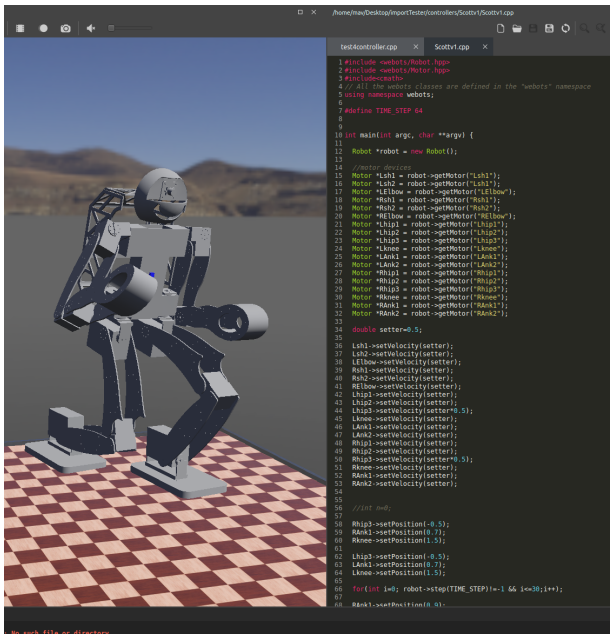

Figure 12: WeBots. Movement animation, also presented in the youtube video.[27]

The second simulation setup is a full ROS-WeBots bridge test. Besides environment, this was used to test and investigate the ROS-WeBots connectivity and the available services, topics and nodes.

0.7.2 Major accomplishments and development cycle

Within the development cycle of various simulation tests, simple objects have been created both using its own creation tools and making very simple mechanical designs in Fusion and exporting them, to see their inherited properties and limitations.

Major accomplishments include; finding out a way to display the default hierarchy representation just like for a robot created with its own tools, allowing for much easier addition of sensors, and editing of the collision meshes. Another major accomplishment was setting up the ROS feedback loop, as it is represents the basis of all future control software tests.

In order to achieve a faster simulation time, WeBots divides the robot into visible meshes (render) and invisible meshes (collision). To compute object collisions, the invisible meshes are used, and they are traditionally set to be either primitives (very simple shapes) or a very simplified model.

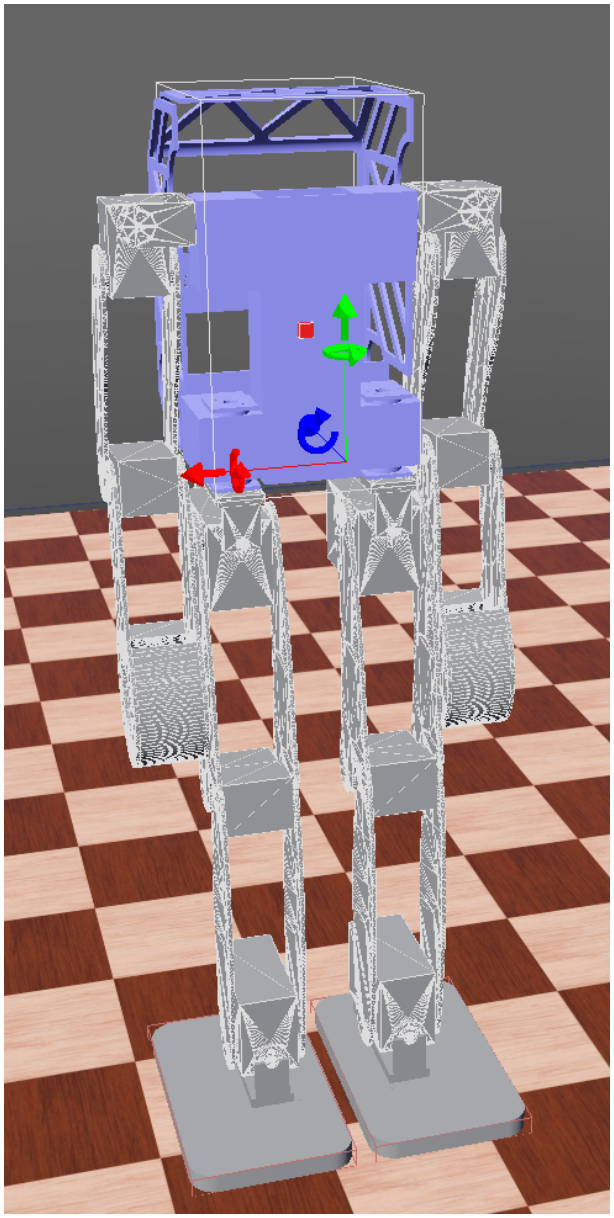

Figure 13: WeBots. Collision meshes shown. Feet and main body, due to complexity, are using cube primitives for the collisions. This can be seen as only outlines instead of full triangulation of the mesh, like with the rest of the links

0.7.3 Current issues and next step

While the community has great presence both on dedicated forums as well as YouTube, and the developers are very active and supportive, there are still some small and very specific topics that are not well covered in the available tutorials, like how to set up a generic ROS - WeBots connection using packages without the terminal. The current project aims to cover some of them.

As the project finishes after completing an upload of all the documentation, materials and generated tutorials, the next step will be handing it to the Robotics Society for the further development of both Scott and any other future humanoids. Part of the handover will be preparing a GitHub with all the 3d files, documentation and manuals that were created or used, and having someone else validate that the work so far can be reproduced.

0.8 Retrospective evaluation of software

While attempting to solve very similar problems, CoppeliaSim and WeBots have great differences which become apparent when used more intensively.Some of the most notable differences are discussed below.

Coordination with other software

CoppeliaSim appears to be heavily based around ROS, as its imported models need to be part of an active ROS project, while WeBots can run fine without it, and has an environment for rapid prototyping of control software inside the application, with live editing and great functionality. WeBots does implement multiple types of ROS bridges,suitable for all operating systems, and the developed packages show the great extent to which even the default controller provides functionality. Both of the software can also use stand-alone scripts for the controllers.

Set up complexity

Undoubtedly, setting up WeBots is significantly less complex than CoppeliaSim. This applies in both with and without the ROS connection. This is because, while CoppeliaSim is installed by simply downloading, WeBots is using an installer that connects to ROS directly and sets up tutorial project spaces. To start using WeBots, starting any tutorial from the application is all that is needed. To start CoppeliaSim with ROS, additional steps are required, which as stated before, don't work in some scenarios at all. To start CoppeliaSim without ROS, using custom controllers, still requires understanding how their package system works, and writing your own package. In WeBots, writing your own controllers is taught in one of the beginner tutorials, and involves a couple clicks in the application and creating a hello world-type file in the editor within the application.

Efficient use of resources

There are studies showing that WeBots has a much more efficient use of CPU and RAM [21], but it also uses the GPU for rendering of both the scene, and the sensor data, even providing live feedback like the camera feed. CoppeliaSim currently doesn't take advantage of the GPU at all.

Community and developer team

While CoppeliaSim used to have a more active research community, currently there are very few up to date tutorials on using it. Its developer team now consists of a single active member. WeBots on the other hand has enough developers for a couple of them to be ever present on Discord and forums, ready to help. Its Youtube community has been increasingly growing in the past year, and the tutorials it documents and comes set up with cover a very large array of cases. Moreover, it also comes with a large array of up to date robots, including humanoids like Atlas.

This may be because of a recent team up between WeBots and RoboCup. Due to Covid, RoboCup currently is holding all the humanoid competitions virtually, which are all taking place with the use of WeBots.

On a professional level, as seen on their homepage, WeBots has a number of impressive projects and collaborations with reputable names in both the industry -Samsung, Thales, Renault- and academia -MIT, Stanford, Tokyo University, Zurich University-.

Physics

CoppeliaSim is able to use multiple physics engines[8]: ODE, Bullet, Vortex and Newton Dynamics, but is designed around simulating the kinematics. WeBots however focuses much more on the physics despite using only one engine: ODE. However it does provide easy support to modify the physics computations, and even rewriting the rules from scratch. [22]

0.9 Logbook and supplementary work

0.9.1 Logbook and work replication

Within the linked logbook[26], a record of meetings, work sessions, research and meaningful notes has been kept.

Inside the notebook, in a separate section titled "Work Replication", a set of small guides has been recorded, as step-by-step instructions of replicating the entire work of the project, using the most efficient methods found or created.

To ensure the relevance of it, the guides have been used multiple times along to project to make fresh setups of all the software, on multiple devices.

0.9.2 TAROS paper

Part of the project, a TAROS paper had been developed under the working title of "A comparison of Humanoid Robot Simulators - a user experience", on the topic of an undergraduate/non power-user point of view of using CoppeliaSim vs WeBots and their general viability and use. This has been developed with great help from my mentor, under the approval of the supervisor, and is due for submission later this month.

A great deal of experience came from writing the paper. There are a number of differences in approaching a paper writing task. The purpose of the paper is to document the results, not the process, so it will skip over all the details of the failed work. The presentation style is for a concerned and up-to-date audience, meaning there is no need or space for explanations of how software works or what each thing does. Finally and probably most importantly, the focus is on how somebody else can use what you have done, not on proving what you have done.

0.9.3 Tutorial, guides and documentation

A tutorial has already been uploaded to Youtube[27] publicly, detailing how to export a robot from Fusion and set it up in WeBots easily, which will likely be remastered, and another will likely follow, providing a comprehensive guide for undergraduates to reproduce the successful work of the project[26].

0.10 Conclusions

In spite of various hindrances, the current project has achieved the initial 4 milestones, without getting to touch on the initial bonuses, but managing to achieve significant other aims. The current report presents the creation of a digital twin of a real world robot (Scott), as well as the set-up of relevant physics and kinematics simulations(WeBots), while bridging the control to a well known dedicated robot control and control simulation software (ROS).

The project underwent a severe modification in the middle of development, and as a result, has highlighted some major issues with the current situation of robot simulators, among which is the great time involvement needed and relatively high knowledge barrier required for approaching such a software. Issues which have been addressed.

A great deal of research work was completed, in order to produce not just well documented guides for setting up a simulator for the future Plymouth undergraduates, but to also publish a paper with potential value to the wider academic community.

0.11 References

[1] UPSU Robotics Society Homepage, https://www.upsu.com/societies/robotics/

[2] Plymouth University Robot Football Homepage, https://www.plymouth.ac.uk/schools/school-of-engineering-computing-and-mathematics/electronics-robotics/robot-football

[3] Plymouth University Robot Football Facebook page, https://www.facebook.com/University-of-Plymouth-Robot-Football-148285347160/

[4] ROS Homepage, https://www.ros.org/

[5] ROS Wiki Homepage, http://wiki.ros.org/

[6] CoppeliaSim Homepage, https://www.coppeliarobotics.com/

[7] CoppeliaSim Acknowledgements page, https://www.coppeliarobotics.com/helpFiles/en/acknowledgments.htm

[8] CoppeliaSim Dynamics Module page, https://www.coppeliarobotics.com/helpFiles/en/dynamicsModule.htm

[9] WeBots Homepage, https://cyberbotics.com/

[10] On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward. HeeSun Choi, Cindy Crump, Christian Duriez, Asher Elmquist, Gregory Hager, David Han, Frank Hearl, Jessica Hodgins, Abhinandan Jain, Frederick Leve, Chen Li, Franziska Meier, Dan Negrut, Ludovic Righetti, Alberto Rodriguez, Jie Tan, Jeff Trinkle; Proceedings of the National Academy of Sciences Jan 2021, 118 (1) e1907856118; DOI: 10.1073/pnas.1907856118 https://www.pnas.org/content/118/1/e1907856118

[11] A Study on the Challenges of Using Robotics Simulators for Testing, Afsoon Afzal, Deborah S. Katz, Claire Le Goues, Christopher S. Timperley, 2020, Carnegie Mellon University, arXiv. https://arxiv.org/pdf/2004.07368.pdf This is well presented in this article: https://www.therobotreport.com/10-challenges-simulators-robotics-testing/

[12] Simulators in Educational Robotics: A Review. Tselegkaridis, S.; Sapounidis, T. Educ. Sci. 2021,https://www.mdpi.com/2227-7102/11/1/11/htm https://doi.org/10.3390/educsci11010011

[13] Fusion to URDF script, 2021, by Florian Fischer "SpaceMaster85", branched from Toshinori Kitamura's original project,https://github.com/SpaceMaster85/fusion2urdf

[14] Fusion to URDF script, 2020, Toshinori Kitamura "Syuntoku14", no longer maintained. https://github.com/syuntoku14/fusion2urdf

[15] URDF to WeBots (PROTO) script, 2021, WeBots, https://github.com/cyberbotics/urdf2webots

[16] Robot State Publisher, ROS package, http://wiki.ros.org/robot_state_publisher

[17] Joint State publisher, ROS package, http://wiki.ros.org/joint_state_publisher

[18] TF2, ROS package, https://wiki.ros.org/tf2

[19] CoppeliaSim official ROS Tutorial, https://www.coppeliarobotics.com/helpFiles/en/ros1Tutorial.htm

[20] CoppeliaSim ROS interface plugin, https://github.com/CoppeliaRobotics/simExtROS

[21] A Comparison of Humanoid Robot Simulators: A Quantitative Approach,Angel Ayala, Francisco Cruz, Diego Campos, Rodrigo Rubio, Bruno Fernandes, Richard Dazeley, August 2020, arXiv and IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob), urlhttps://arxiv.org/abs/2008.04627

[22] WeBots (official) Tutorial, introduction to physics plugins, https://www.cyberbotics.com/doc/reference/introduction-of-the-physics-plugins

[23] RoboCup News page, Call for Participation: RoboCup Worldwide 2021 Virtual Humanoid League, https://www.robocup.org/news/108

[24] PID controller Wikipedia page, https://en.wikipedia.org/wiki/PID_controller

[25] Kitware APT Repository homepage, https://apt.kitware.com/

[26] The GitHub page that hosts the entire project documentation, including robot model, https://github.com/Daemongear/RobotSimulator

[27] Youtube project presentation, https://www.youtube.com/watch?v=zl5T3P4wZOk

0.12 Bibliography

0.13 Appendices

0.13.1 WeBots ROS controller services for a single motor

This is a full list for the motor "RAnk2", taken directly from the terminal feedback.

```
/Assembly2v2_2829_ubuntu/RAnk2/get_acceleration
/Assembly2v2_2829_ubuntu/RAnk2/get_available_torque
/Assembly2v2_2829_ubuntu/RAnk2/get_brake_name
/Assembly2v2_2829_ubuntu/RAnk2/get_max_position
/Assembly2v2_2829_ubuntu/RAnk2/get_max_torque
/Assembly2v2_2829_ubuntu/RAnk2/get_max_velocity
/Assembly2v2_2829_ubuntu/RAnk2/get_min_position
/Assembly2v2_2829_ubuntu/RAnk2/get_model
/Assembly2v2_2829_ubuntu/RAnk2/get_name
/Assembly2v2_2829_ubuntu/RAnk2/get_node_type
/Assembly2v2_2829_ubuntu/RAnk2/get_position_sensor_name
/Assembly2v2_2829_ubuntu/RAnk2/get_target_position
/Assembly2v2_2829_ubuntu/RAnk2/get_type
/Assembly2v2_2829_ubuntu/RAnk2/get_velocity
/Assembly2v2_2829_ubuntu/RAnk2/set_acceleration
/Assembly2v2_2829_ubuntu/RAnk2/set_available_torque
/Assembly2v2_2829_ubuntu/RAnk2/set_control_pid
/Assembly2v2_2829_ubuntu/RAnk2/set_position
/Assembly2v2_2829_ubuntu/RAnk2/set_torque
/Assembly2v2_2829_ubuntu/RAnk2/set_velocity
/Assembly2v2_2829_ubuntu/RAnk2/torque_feedback_sensor/enable
/Assembly2v2_2829_ubuntu/RAnk2/torque_feedback_sensor/get_sampling_period
/Assembly2v2_2829_ubuntu/RAnk2_sensor/enable
/Assembly2v2_2829_ubuntu/RAnk2_sensor/get_brake_name
/Assembly2v2_2829_ubuntu/RAnk2_sensor/get_model
/Assembly2v2_2829_ubuntu/RAnk2_sensor/get_motor_name
/Assembly2v2_2829_ubuntu/RAnk2_sensor/get_name
/Assembly2v2_2829_ubuntu/RAnk2_sensor/get_node_type
/Assembly2v2_2829_ubuntu/RAnk2_sensor/get_sampling_period
/Assembly2v2_2829_ubuntu/RAnk2_sensor/get_type
```

Other available services are:

```
/Scott3_3082_ubuntu/battery_sensor/enable
/Scott3_3082_ubuntu/battery_sensor/get_sampling_period
/Scott3_3082_ubuntu/get_loggers
/Scott3_3082_ubuntu/inertial_unit/enable
/Scott3_3082_ubuntu/inertial_unit/get_model
/Scott3_3082_ubuntu/inertial_unit/get_name
/Scott3_3082_ubuntu/inertial_unit/get_node_type
/Scott3_3082_ubuntu/inertial_unit/get_noise
/Scott3_3082_ubuntu/inertial_unit/get_sampling_period
/Scott3_3082_ubuntu/joystick/enable
/Scott3_3082_ubuntu/joystick/get_model
/Scott3_3082_ubuntu/joystick/get_number_of_axes
/Scott3_3082_ubuntu/joystick/get_number_of_povs
/Scott3_3082_ubuntu/joystick/get_sampling_period
/Scott3_3082_ubuntu/joystick/is_connected
/Scott3_3082_ubuntu/joystick/set_auto_centering_gain
/Scott3_3082_ubuntu/joystick/set_constant_force
/Scott3_3082_ubuntu/joystick/set_constant_force_duration
/Scott3_3082_ubuntu/joystick/set_force_axis
/Scott3_3082_ubuntu/joystick/set_resistance_gain
/Scott3_3082_ubuntu/keyboard/enable
/Scott3_3082_ubuntu/keyboard/get_sampling_period
/Scott3_3082_ubuntu/robot/get_basic_time_step
/Scott3_3082_ubuntu/robot/get_custom_data
/Scott3_3082_ubuntu/robot/get_data
```

```
/Scott3_3082_ubuntu/robot/get_device_list
/Scott3_3082_ubuntu/robot/get_mode
/Scott3_3082_ubuntu/robot/get_model
/Scott3_3082_ubuntu/robot/get_number_of_devices
/Scott3_3082_ubuntu/robot/get_project_path
/Scott3_3082_ubuntu/robot/get_supervisor
/Scott3_3082_ubuntu/robot/get_synchronization
/Scott3_3082_ubuntu/robot/get_time
/Scott3_3082_ubuntu/robot/get_type
/Scott3_3082_ubuntu/robot/get_urdf
```