FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Simulated Assembly of Flexible Cabling Systems for Vehicles

David Freitas Dinis

Mestrado em Engenharia Informática e Computação

Supervisor: Professor Armando Jorge Miranda de Sousa

Second Supervisor: Gonçalo da Mota Laranjeira Torres Leão, MSc.

July 30, 2022

Simulated Assembly of Flexible Cabling Systems for Vehicles

David Freitas Dinis

Mestrado em Engenharia Informática e Computação

July 30, 2022

Abstract

Automotive cabling systems are quite challenging to assemble and that takes manufacturing companies a considerable amount of time that could be saved with automatic cable placement. In particular, in the automotive industry, each cable is being placed by hand due to the extensive number of variables the cables have. With the more advanced features of cars nowadays, the length and complexity of the cables also increases and has become a real problem as the cars today have over 4 km of cable compared to a few hundred meters 30 years ago.

Automatic systems capable of tackling this problem must encompass various components such as grasp planning and motion planning, with friction and cable flexibility in mind. Each of these components has its limitations and must work together to fit each cable at a time and obey the industry specifications.

The goal of this dissertation is to model the cable, the mounting jig and the robot arm in a simulation environment in Multi-Joint dynamics with Contact (MuJoCo). Then, a strategy was developed to mount all the cables with the robot arm. Each end point of the cable must be assembled to the connector. Between each end point assembly, the cable is passed by each fork and as such assembled from right to left.

The performance of the system was measured by the average time spent and the number of times that the cable passed in the fork at each run. To improve knowledge of the parameters of the system, experiments were made with a unique configuration each. To compare the performance of these experiments, a baseline experiment were created. The experiments only changed one parameter from the baseline. The baseline consisted of all random parameters (fork positions and fork rotations) and had 4 forks. To create realism, an experience where each fork is rotated toward the fork that is immediately at the right was made. This experiment provides the best results, with a cable-passing-correctly-through-the-fork success rate of 87.5%, despite the simplicity of the wire assemble algorithm. The experiment had an average time of 87.47 seconds and with a standard deviation of 6.27 seconds to pass the cable along four forks and the two connector endpoints. It is also important to mention that the jig table has dimensions of 1 meter by 70 centimeters by 5 centimeters, and the wire had a length of 2.5 meters.

Keywords: Cable manipulation, Deformable Linear Objects, Industrial Robots, Motion Planning, Path Planning, Simulation

Resumo

A construção de cablagens automóveis é uma tarefa bastante desafiante que leva às fábricas uma quantidade considerável de tempo que poderia ser poupada com a automatização da colocação dos cabos. Em particular, na indústria automóvel, cada cabo deve ser construído à mão devido ao grande número de variáveis que os cabos têm. Com as capacidades mais avançadas dos carros atuais, o comprimento e a complexidade dos cabos aumentou e tornou-se um problema sério, visto que os carros de hoje têm mais de 4 km de cabo em comparação com algumas centenas de metros há 30 anos.

Sistemas capazes de enfrentar este problema devem abranger vários componentes, tais como planeamento de agarrar e planeamento de movimento, com fricção e flexibilidade de cabos em mente. Cada um destes componentes tem as suas limitações e deve trabalhar em conjunto para encaixar um cabo de cada vez e obedecer às especificações da indústria.

O objetivo desta dissertação é modelar todos os cabos, o suporte de montagem e o manipulador robótico num ambiente de simulação em Multi-Joint dynamics with Contact (MuJoCo). Depois, foi desenvolvida uma estratégia para montar todos os cabos com o manipulador robótico, isto é, fazer passar o cabo por todos as forquilhas de montagem. Cada ponta do cabo terá que ser montada no conector. Entre a montagem das pontas, as forquilhas são montadas da direita para a esquerda.

O desempenho do sistema foi medido pelo tempo médio gasto e pelo número de forquilhas montadas com sucesso de cada execução. Para melhorar o conhecimento dos parâmetros do sistema, foram feitas diversas experiências com uma configuração única. Para comprar o desempenho destas experiências, foi criada uma experiência base. As experiências alteram apenas um parâmetro a partir da experiência base. A experiência base consiste em quatro forquilhas parâmetros aleatórios (posições e rotação). Para simular o mundo real, foi feita uma experiência em que cada forquilha está virada para a forquilha imediatamente à direita. Esta experiência proporcionou os melhores resultados, com uma taxa de sucesso de passagem do cabo nas forquilhas de 87,5%, apesar da simplicidade do algoritmo de montagem do cabo. A experiência teve um tempo médio de 87,47 segundos e com um desvio padrão de 6,27 segundos para fazer passar o fio em quatro forquilhas e colocação nos conectores das extremidades do cabo. Também é importante mencionar que a mesa de montagem tem dimensões de 1 metro por 70 centímetros por 5 centímetros e o cabo tem um comprimento de 2.5 metros.

Keywords: Manipulação de Cabos, Objetos Lineares Deformáveis, Planeamento de Movimento, Planeamento de trajectos, Robots industriais, Simulação

Acknowledgements

To begin with, I would like to thank my supervisor, Professor Armando Sousa, for the guidance and the ideas that gave purpose to this dissertation.

Also, it is long overdue to acknowledge my second supervisor, Gonçalo Leão, for the patience and help with the ROS software. He was always available to talk and pull the work forward.

Last, I would like to thank my friends and family that supported me and were always there.

David Dinis

"You have to work hard for envy.
You get pity for free."

Günther Steiner

Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
DLO	Deformable Linear Object
MuJoCo	Multi-Joint dynamics with Contact
RL	Reinforcement Learning
RRT	Rapidly-exploring Random Tree
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language

Chapter 1

Introduction

1.1 Context

Robot systems came a long way since 1954, where the first industrial robot was made by a company named Unimation Inc. This robot was first implemented by General Motors Corporation in a die-casting factory. The first production line robot was implemented in 1961 by the same company [Moravec, 2021].

Vehicles today have an increasing demand for technology as society grows further. This increasing demand goes hand in hand with heavier cars and needs for more complex systems. This evolution of embedded systems requires an increase of wires and connectors, which results in longer production times. In this dissertation, tubes and wires and used interchangeably.

The assembly of flexible wires takes many steps and is a tedious process. With robot arms, this process can be automated, resulting in faster assembly and freeing people of the burden of repetitiveness.

Bruce Buchanan describes that the idea of AI began with science fiction and philosophers [Buchanan, 2005]. At the time, researchers were limited by the technology available, and so only in the last century it could be done. Today, AI is advanced enough to beat humans in many tasks and is of great utility to mankind. There are various approaches to use AI to accomplish a task.

1.2 Motivation

Companies can benefit greatly from the automation that robot systems can provide. This system can automate processes, making them faster and more efficient. With such system, the companies can make a more competitive price and such close the gap between Large Enterprises and Small and Medium-sized Enterprises (SME).

Deformable Linear Objects (DLO) assembly is an essential part of the car industry. This process can take long and the industry would benefit greatly from the production of a robot system that can do this job. The vehicle wire-harnesses have bifurcations, which greatly increase the difficulty of manipulation and planning. So, this process is still a manual job that can have safety

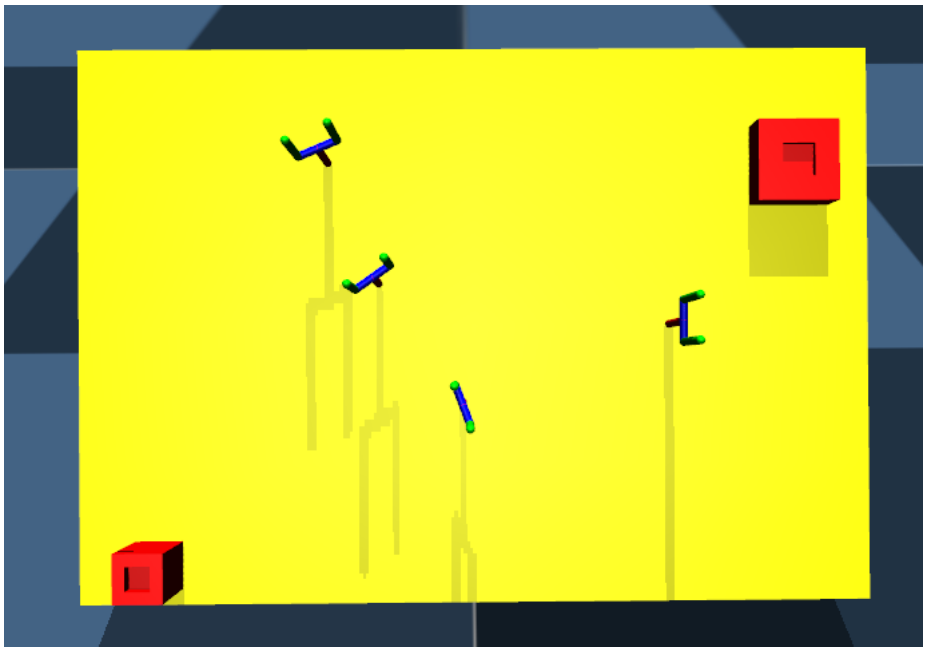


Figure 1.1: Jig not assembled

and health consequences, as the workers perform repetitive tasks that can contribute to physical as well as mental problem of the employees.

1.3 Objectives

This dissertation focus on creating a simulated Robotic Operating System (ROS) environment that is easy to work and can provide with a simple to implement way to move the wires. Since there are no correct sequence of events to accomplish this goal, it turns the task of automation hard. Although there is a set of rules that the model has to obey and so using a heuristic could be of interest to get a solution. In Figure 1.1 it is possible to see the jig yet to be assembled.

Another important part of the system that is required, is the models of the assembly jig and the wires. This will be done by parsing the parameters of the scene into the MuJoCo engine.

1.4 Research Questions

RQ1. How can an assembly jig be modeled in simulation?

RQ2. What is an adequately simple strategy to mount a flexible wire in an assembly jig?

1.5 Document structure

The rest of the document has 6 parts. Chapter 2 describes the state-of-the-art of the manipulation of cables and wires using robotic systems. Chapter 3 presents the relevant tools that were used in

this dissertation. Chapter 4 exhibits how the work done in this dissertation was done. Chapter 5 shows the results taken with different experiments. Chapter 6 presents the conclusions.

Chapter 2

Fundamentals of Deformable Linear Objects (DLO)

In the survey [Sanchez et al., 2018], a deformable object is defined as an object that has no compression strength or have a large strain or have a large displacement as seen in figure 2.1. Compression strength occurs when 2 opposite ends are pushed against each other and the object offers resistance. It is important to know the difference between uniparametric, biparametric and triparametric. Uniparametric objects have 1 dimension significantly larger than the other 2, one example of this is a rope or a wire. Biparametric objects have 1 dimension much smaller than the other 2. Triparametric objects are the regular solid objects. With these definitions, we can divide objects by 4 main categories:

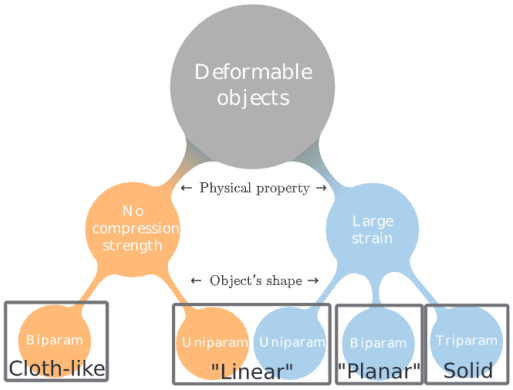


Figure 2.1: Classification of deformable objects [Sanchez et al., 2018]

- **Linear** - Do not present compression strength and are uniparametric
- **Planar** - Biparametric with large displacement
- **Cloth-like** - Biparametric that do not present compression strength
- **Solid or Volumetric** - Triparametric objects

Although the manipulation of rigid objects is a well studied field, it is not possible to apply the same principles to deformable objects especially to deformable linear objects like cables and ropes due to their large amount of degrees of freedom.

2.1 Representation of DLO's

2.1.1 Data acquisition

The manipulation of a DLO starts with getting information about their geometrical properties. The most common solution for this problem is equipping the robotic system with a camera or with a tactile system in the grippers. An example of a gripper with a tactile sensor can be seen in figure 2.2.

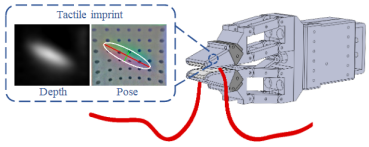


Figure 2.2: Example of a gripper with a tactile sensor [She et al., 2019]

GelSight is a vision-based tactile sensor that can detect the pose of the cable when gripped and the friction values when the cable is sliding. This technology enables to grip a cable in 2 points and slide one of the grippers until the end of the cable as we can see in figure 2.4(a) [She et al., 2019].

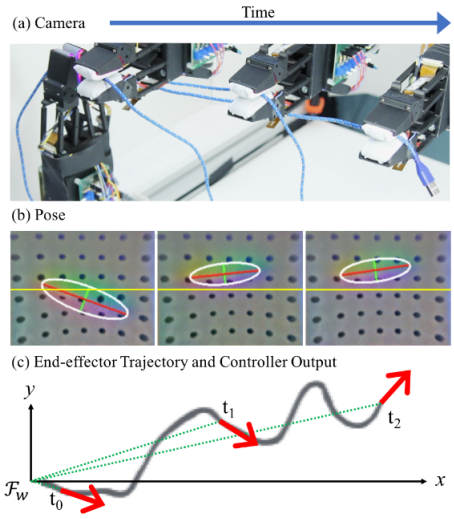


Figure 2.3: Example of a gripper with a tactile sensor [She et al., 2019]

Camera systems are far more common within the robotic community due to the easy accessibility. In order to improve the estimated 3D position of the cables, a stereo vision system would

be the best use of the cameras. This solution has its limitation due to vision occlusions [mo Koo et al., 2008].

These vision systems could also be implemented with deep learning. Using both vision and tactile systems can improve the overall performance of the manipulation of the cable. Using the camera with deep learning to detect where the cable is and grasp it. Once grasped, a tactile sensor could regrasp the cable if the position is not the desired, this could happen due to visual occlusions [Caporali et al., 2021].

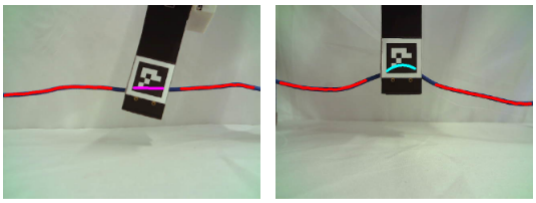


Figure 2.4: Vision and tactile representation with linear tactile map interpolation (a) and with quadratic tactile map interpolation(b) [Caporali et al., 2021]

2.1.2 Topological vs geometric models

Sometimes the exact shape of the cable is not so important for some applications and the intersections with itself or the environment is a far better way of representing the DLO. With knot tying, the cross sections is what defines the knot and so is the logical way of representing the cable as seen in [Saha and Isto, 2007]. Therefore, the topological representation is a vector that each element represents and intersection of the cables. The elements are composed with 3 values:

$$(X_j^1, X_j^2, \varepsilon_j)_{j=1,\ldots,n} \tag{2.1}$$

- X_j^1 - distance of the first point to the beginning of the cable
- X_j^2 - distance of the second point to the beginning of the cable
- ε_j - is +1 if the counterclockwise angle between the top cable and the under cable is less than π or if the clockwise angle between the under cable and the top cable and less than π otherwise is -1.

  This is great for knot tying, but once there are several cables that cannot be twisted it is not the better approach. Lv et al. solved this problem by creating a cable model that takes into account the twisting deformations as well as the contacts, collisions, friction and the kinematic constrains of both ends of the DLOs [Lv et al., 2022].

2.1.3 State Estimation

If the cable is grasped at both ends and the length is known, it is possible to predict the shape. The predicted shape only refers to the cable inside the grasping points, the "interior" solution.

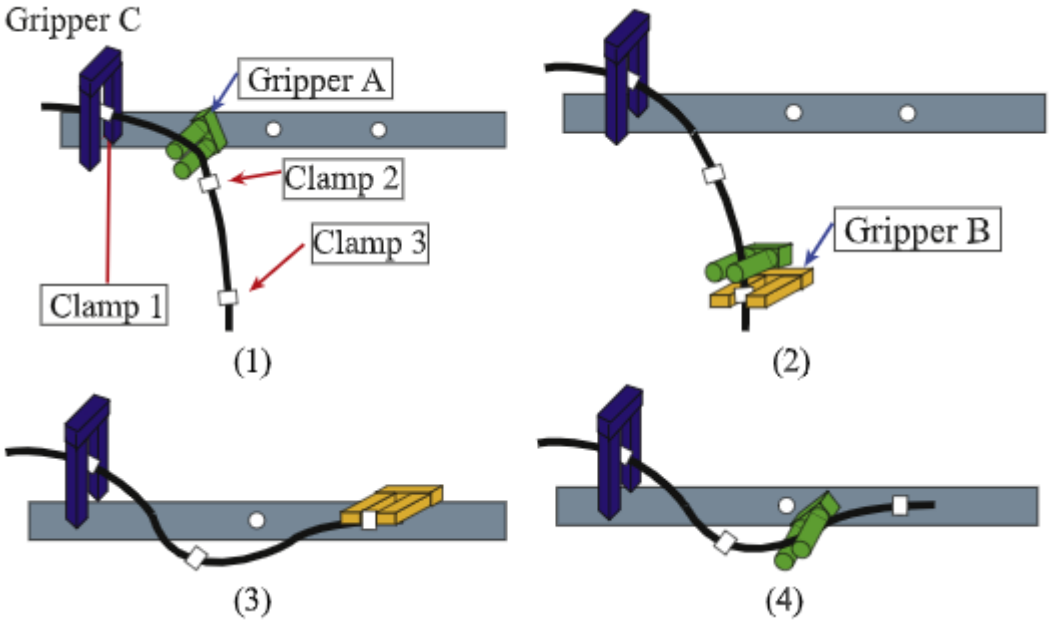


Figure 2.5: Wire tracing motion [Jiang et al., 2015]

This interior solution can be defined using a catenary curve with only 1 free parameter which corresponds to the length of the cable between the grasping points. A Gaussian quadrature approximation of 5 points followed by a minimization can solve the length problem. In order to simplify the computational complexity, the shape of the cable is represented by a quadratic function [Papacharalampopoulos et al., 2015].

If the cables have specific grasping points and the length of the subdivision is known, the shape can be computed. With this method the entire shape of the cable can be determined by processing each subdivision separately [Shah and Shah, 2016]. As an alternative, Jiang et al. propose a tracing algorithm that tries to mimic what a human would do if they were asked to do the task blindly [Jiang et al., 2015]. To accomplish this, there should be at least 2 robotic grippers. One should hold an end point of a wire, and with the other, grip the wire as closely as possible to the first gripper. With this setup it should be easy to slide the second gripper until it reached and connector or the other end point as seen if figure 2.5.

When dealing with simulation, there should be a balance between performance and realism. In [Moll and Kavraki, 2006] Moll and Kravaki propose a solution to this using minimal-energy curves to represent the cable. The algorithm calculates the paths from one minimal-energy curve to another, and thus the intermediate curves are also minimal-curves. Veltkamp and Wessenlink in [Veltkamp and Wesselink, 1995] describe this minimal-energy curve as a curve that minimizes a number of interpolation constraints. This results in an overall smooth curve. This solution limits the degrees of freedom that the DLO has. Earlier, Moll and Kavraki [Moll and Kavraki, 2004] managed to get to 10 degrees of motion using this method.

2.2 Motion Planning

In order to mount the cables properly, a strategy is required and is what defines the motion planning problem. There are several methods of accomplishing the motion planning problem. The methods are mainly divided in two categories.

2.2.1 Heuristic methods

Heuristic methods are popular when there are complex information and could have more than one solution. This approach to problem-solving uses a practical method that may not provide the best result, but is sufficient given the time and resources they take.

To achieve a desired shape of the cable, both ends could be manipulated to get the desired effect. Zhu et al. propose using a multi-arm robot to grasp the cable at both ends and has 3 degrees of freedom. It can move in the X and Y axes, as well as rotate in the Z axis [Zhu et al., 2018]. With this setup, it was shown that it can manipulate the cable to accommodate a specified shape, as shown in figure 2.6.

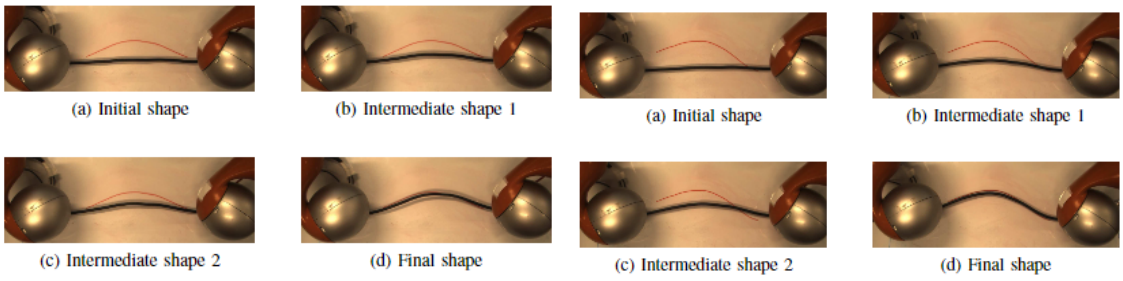


Figure 2.6: Two examples of manipulating the cable to the specified shape [Zhu et al., 2018]

Some cables have interlinks that are fragile and cannot withstand the weight of the cable. So the cable cannot be manipulated by these interlinks without risking damaging their electrical integrity. In order to manipulate the cable, 5 action types were defined:

- **RepositionManipulator** - Reposition a specific manipulator to a specified location
- **GraspCable** - The selected manipulator grasps the cable at the specified grasping point
- **ClampCable** - The selected manipulator moves the cable to the clamp specified and tranfers the boundary value to the clamp
- **ReleaseManipulator** - The selected manipulator lets go of the cable
- **AlighnReferencePoint** - This is a compound move that given a reference point the closes grasping point is selected, and the specified manipulator uses the GraspCable action followed by the ClampCable.

With these actions, a manipulation plan may be built using action sets. The action sets are tuples of action that when together form a complex action [Shah and Shah, 2016].

This solution may not lead to great success in constrained environments, since it does not consider collisions by the cable and the surroundings. Roussel and Taïx [Roussel and Taïx, 2014] solved this problem by coupling dynamic simulation for the DLO and kinodynamic motion planning with contacts. To handle with the kinodynamic constrains, a Rapidly-exploring Random Tree (RRT) could be used. These proved to be a good solution to the collision problem. [Lamiraux and Kavraki, 2001] also tackles this problem by deforming the manipulated object in order to not collide with the environment using the principles of elasticity theory of mechanics.

2.2.2 Machine Learning

Machine learning is a subcategory of AI where the computer can learn how to do a task without the need to be programmed. Within the machine learning methods, there are various categories [1]:

- **Supervised Learning** - This method is the most appropriate when labeled data is available. The algorithm tries to infer the correlation of the data with the corresponding label. The problems that supervised learning focuses on are classification and regression. Classification has discrete labels that are distinct from each other. Regression, on the other hand, focus on predicting a continuous variable.
- **Unsupervised Learning** - It is not always possible to have labeled data, so this method focuses on trying to discover patterns and group the data in clusters.
- **Reinforcement Learning** - It specializes in achieving a goal defined using a reward system. When the machine performs an action that is desirable, it gets rewarded. Then the main goal is to maximize the rewards obtained. This can be extended to remove rewards when the machine does a not so desirable action.

2.2.2.1 Reinforcement learning

Usually, learning-based approaches to the motion planning problem require large amounts of simulated datasets or human demonstrations, or even both. Wu et al. [Wu et al., 2019] tackle this problem using model-free visual reinforcement learning (RL). To improve the efficiency of the learning procedure, they made two changes to the more usual RL methods. The first one is to implement an iterative pick-place action space, and so the robot could decide where to pick and where to drop. This could lead to so problems since the optimal placing point depends on the picking point. The other one is to separate the learning of picking and placing. To do this, the placing policy is only learned by conditioning random picking points. the picking policy is then defined by selection the picking point that has the maximal value under placing and solving the problem of using an iterative pick-place action space. In figure 2.7 the main differences of the implemented policies can be seen.

[1] https://towardsdatascience.com/machine-learning-basics-part-1-a36d38c7916

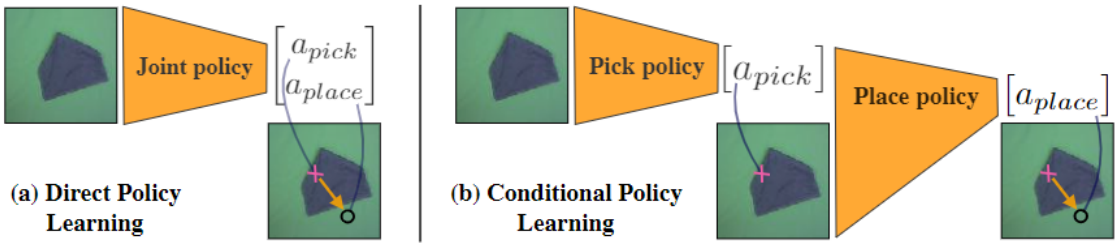


Figure 2.7: Differences between normal RL policies and the solution in [Wu et al., 2019]

2.2.2.2 Supervised Learning

Lee et al. presented an approach to this that can achieve its goal with only 1000 samples of real-world data in [Lee et al., 2022]. In order to achieve these results, they used a fully-convolutional neural network with the input and output image transformed into a canonical representation in order to reduce the complexity of the network architecture and to avoid fully-connected layers. They combine this process with a self-supervised data collection process where the robot resets the DLO and perform random pick-and-place actions at regular intervals. Mahler et al. approached this problem by creating a point cloud to represent the objects. with the point cloud database it is possible to train a Grasp Quality Convolutional Neural Network (GQ-CNN) that can predict the success rate of a grasp from those point clouds [Mahler et al., 2017].

2.3 Assembly of flexible wires for vehicles

In order to determine if the problem being done in this dissertation is relevant and brings some contributions to the community, a query was made on February 20, 2022, to some relevant research databases. The query was composed by 3 sub queries that aimed to find:

- Publications where objects are wires or cables and related to vehicles;
- Publications that used robotic systems;
- Publications where objects are grasped, handled or manipulated.

Table 2.1: Assembly of flexible wires for vehicles queries

Research database	**Query**	**No. results**	**No. relevant results**
IEEE Xplorer	((cabl* OR wire*) NEAR/5 (vehic*)) AND (robot*) AND (grasp* OR handl* OR manip*)	132	1 [mo Koo et al., 2008]
Scopus	TITLE-ABS-KEY ((cabl* OR wire*) W/5 (vehic*)) AND TITLE-ABS-KEY (robot*) AND TITLE-ABS-KEY (grasp* OR handl* OR manip*)	51	2 [Shi et al., 2012] [Jiang et al., 2011]
Engineering Village	((cabl* NEAR/5 vehic*) OR (wire* NEAR/5 vehic*)) AND (robot*) AND (grasp* OR handl* OR manip*)	311	1 [mo Koo et al., 2008]

A result is considered relevant if there are wires or cables of any kind being dealt by a robotic system. As shown in table 2.1 most of the results were not appropriate for the task at hand. However, there were 3 papers that somewhat resemble what is trying to be done with this dissertation.

- **[mo Koo et al., 2008]** - As described in 2.1.1 The authors used a stereo vision system to calculate the 3D position of the wires and detecting the clamps where the wire should be attached and using a simple motion planning that calculates the trajectories for the robot and assembles the wire in the clamps one by one. Relying on camera systems could be a problem, since if the camera fails to detect some object, the whole operation will fail. This is described in this paper.
- **[Shi et al., 2012]** Combining a manipulator arm that can perform fine assembly tasks, using both visual and force sensors, and mobile tracking capabilities using visual and laser sensors, they managed to complete the task of mounting the cables in the vehicle while moving. Since the vehicles are mobile, there could be significant error when using this approach.
- **[Jiang et al., 2011]** Using 3 robotic arms and a camera vision system of 16 units, 10 fixed and the other 6 mounted in the robotic arms, they managed to successfully assemble the wire-harness into the expected position using simple and straightforward approaches. This work managed a success rate of 50% in their experiments. This is mainly due to the ineffectiveness of the camera system and torsion in the wire-harness, making the solution unfeasible.

With the table 2.1 in mind, a very reduced number of papers deal with the assembly of wire-harnesses for vehicles. Therefore, it can be concluded that the work done in this dissertation can be of value to the state-of-the-art.

2.4 Summary

In this chapter is described the main components of the manipulation deformable of cables using robotic systems. The revision of the state-of-the-art was concluded by a thorough systematic literature review. With that, is possible to conclude that there is very few work done towards the automation of the assembly of wire-harnesses for vehicles.

Chapter 3

Relevant tools

In this chapter is described the relevant tools that were used in this dissertation.

3.1 ROS

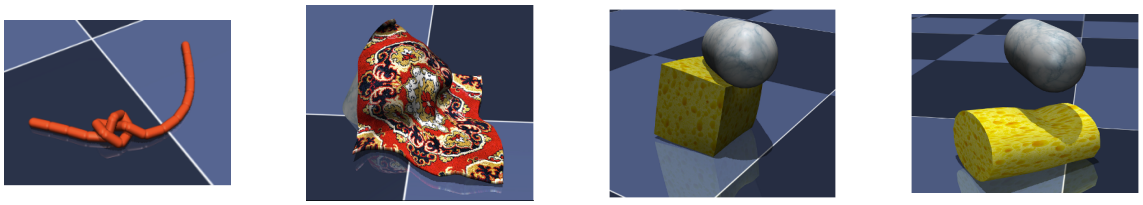
Robotic Operating System (ROS[1]) is an open source set of libraries that help with the development of robotic systems. Although it is called operating system, it is meant to be run on top of an actual operating system. With this in mind it can provide with features that are expected from an operating system:

- hardware abstraction;
- low-level device control;
- message-passage between processes;
- package management.

ROS has many distributions[2] that correspond to a version of the packages. All the work done in this dissertation was done with ROS Melodic Morenia for the legacy code of [Leão, 2020] that is used. Ubuntu 18.04(Bionic) was chosen as the operating system since it is the primary target operating system of Melodic Morenia.

The code developed for ROS is divided into different packages. These packages can be nodes that ROS will execute or ROS-independent code that can be used by the other nodes. This architecture enables the modules to be used in other projects and the nodes to be developed in a modular fashion. ROS is also used vastly in the industry, which increases the relevance of using ROS.

[1] https://www.ros.org
[2] http://wiki.ros.org/Distributions



(a) 1D grid (b) 2D grid (c) Box (d) Cylinder

Figure 3.1: Different types of flexible objects supported by MuJoCo[5]

3.2 MuJoCo

Multi-Joint dynamics with Contact (MuJoCo[3]) is a free and open source physics engine that is designed to facilitate the development of robotic systems. It is written in C/C++ with an Application Programming Interface(API) in C. MuJoCo makes it easy to model all the elements of a scene due to their XML file format that is designed to be as readable as possible.

This Extensible Markup Language(XML) file is parsed by the runtime simulation module that is tuned to maximize performance. At runtime the simulation parameters are divided into 2 data structs:

- mjModel - This data structure is created at compile time and is expected to remain constant;
- mjData - This data structure contains all the dynamic parameters and intermediate results.

With this separation is possible to simulate the same model simultaneously, since the states are separate. The version used was 2.1.1[4].

Mujoco has the ability to model flexible objects such as:

- Rope;
- Cloth;
- Boxes;
- Cylinder and ellipsoid.

These objects, seen in Figure 3.1, are a collection of regular bodies from MuJoCo that come together to create particle systems, ropes, cloth, and soft bodies. These objects can be seen as 1d, 2d or 3d particle systems, that have the particles constrained to move together, and so simulate different flexible objects.

3.3 Summary

This chapter presented the most important software tools that were used in this dissertation, including a baseline for robotic development and a physics engine.

[3] https://mujoco.org
[4] https://github.com/deepmind/mujoco/releases/tag/2.1.1
[5] https://mujoco.readthedocs.io/en/latest/modeling.html

Chapter 4

Proposed solution for the simulated assembly of flexible wires

4.1 Model for the flexible wires and the assembly jig

In order to successfully perform the assembly of the cable in the jig, these must be modeled correctly. To do this, it was separated into two steps: the assembly jig and the wires. Both of the steps are implemented as a ROS package.

4.1.1 Assembly jig

The process begins with loading the parameters that are present in a YAML Ain't Markup Language (YAML) file, seen in Listing 4.1, to ROS. Once these parameters are converted into rosparams the Assembly jig models package can read them, in Figure 4.1 it is possible to see these parameters. The package receives the ROS params and creates the models for the assembly jig in the MuJoCo XML format.

As seen in Figure 4.2, once the models are created, the AssemblyJigTranslator object parses them into XML so MuJoCo can properly display the models.

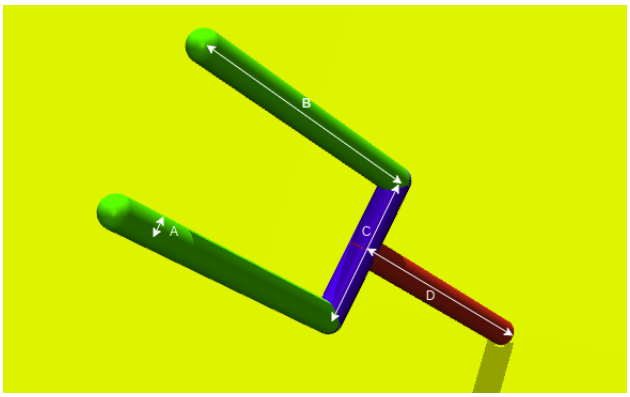


Figure 4.1: Fork parameters. A-cylinder radius, B-prong length, C-distance between prongs, D-neck length

```yaml
assembly_jig:
  jig__properties:
    width: 1
    height: 0.7
    wall_width: 0.05
  a__fork:
    capsule_radius: 0.005
    neck_length: 0.1
    prong_length: 0.1
    prongs_dist: 0.05
    theta: 0.5
    position: [0.2, 0.2]
  b__fork:
    capsule_radius: 0.005
    neck_length: 0.1
    prong_length: 0.1
    prongs_dist: 0.05
    theta: 0
    position: [0.1, -0.1]
  c__fork:
    capsule_radius: 0.005
    neck_length: 0.1
    prong_length: 0.1
    prongs_dist: 0.05
    theta: 0
    position: [-0.1, 0.2]
  d__fork:
    capsule_radius: 0.005
    neck_length: 0.1
    prong_length: 0.1
    prongs_dist: 0.05
    theta: 0
    position: [-0.3, 0.1]
  a__connector:
    depth: 0.05
    width: 0.1
    inner_width: 0.04
    inner_depth: 0.01
    position: [0.4, 0.2]
  b__connector:
    depth: 0.1
    width: 0.06
    inner_width: 0.03
    inner_depth: 0.02
    position: [ -0.4, -0.3 ]
```

Listing 4.1: Asembly jig YAML file

4.1.2 Wires

Once the assembly jig is translated to a cpp structure, the XML file is created and the structures is passed to the assembly jig client described in subsubsection 4.3.3.1. After the assembly jig is processed, the wires structure is created as well as the XML.This process is a simplified version of

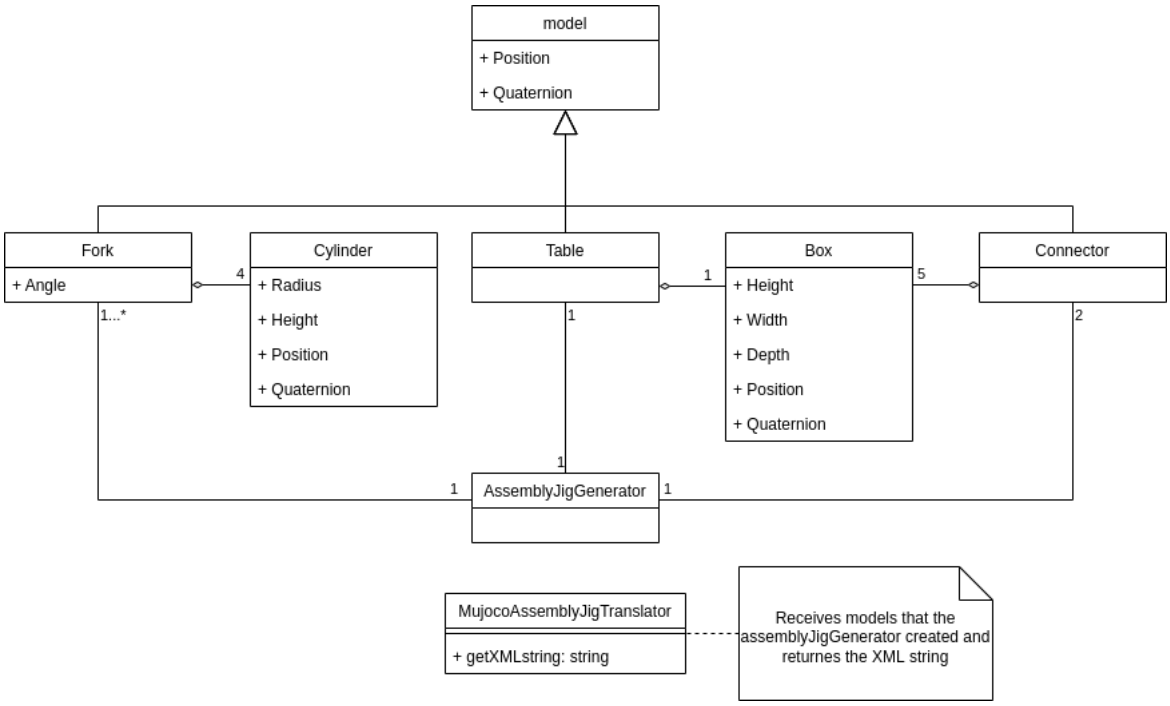


Figure 4.2: UML class diagram for the assembly jig

the algorithm used in [Leão et al., 2022]. This algorithm has a similar solution to the one described in the previous section. First, it reads the YAML file that contains all the variables that are needed to build the wires. Once the variables are read from the ROS params, it creates all the cylinders that are needed in order to simulate a flexible wire with the respective joints between them. The output of the algorithm is an XML file and the wire structure represented by classes. The flexible wire is composed of a set of rigid capsules connected by two hinge joints.

Similarly to [Leão et al., 2020, Leão et al., 2022] with simulating rigid curved tubes in Gazebo, capsules were preferred over cylinders to represent each element of the wire to prevent the object from having gaps throughout its length, which could compromise the simulation's accuracy. These cylinders are connected through two hinge joints. This provides two degrees of motion to each capsule, allowing each capsule to freely rotate with respect to its adjacent neighbours, with the sole constraint of not being able to rotate around the capsule's axis (which would correspond to a torsional deformation). With this in mind, MuJoCo uses a particle-based model based on a mass-spring system for the hinge joints, and thus emulating the wire's dynamics.

4.2 Architecture overview

When the creation of the models finished, the part of the algorithm that controls the movement of the wires can begin. This process begins by notifying the weld action server of both ends of the wires as well as the connectors, seen in Figure 4.3 as the weld bodies, so when they are close enough to be welded it can do so.

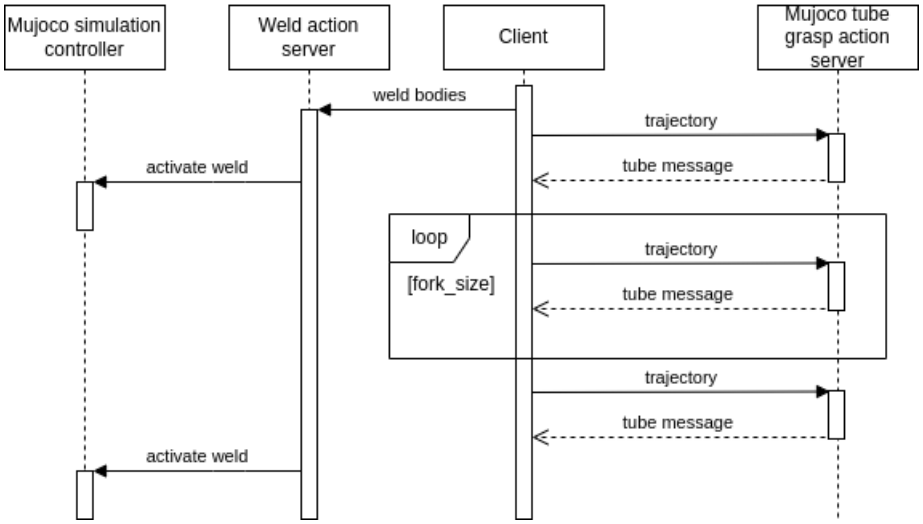


Figure 4.3: UML sequence diagram for the system

The wire end is grasped and moved to the first connector. Once this process is done, the wire must pass through each fork. With this in mind, the cylinder that will be placed in the fork can be grasped. This cylinder can be determined by the sum of the distance of the forks and its previous fork until we reach the connector.

The selected cylinder, once grasped, is moved to the first position, as seen in Figure 4.5, followed by the second position. This part of the algorithm is executed for each fork.

After all the forks have been assembled, the only thing missing is the last connector. In similarly to the assembly of the first connector, the other end of the wire is grasped and moved to the center point of the connector.

4.3 Packages

This section describes the architecture of the dissertation work. The code is separated into two main parts: the server and the client. The server and client are built on top of the simple_action_server library that can be found in the actionlib package of ROS. To communicate between the servers and the client the messages must be defined with a .action file. This file is composed of three definitions:

- Goal;
- Result;
- Feedback.

This file works as an interaction point between the servers and the clients.

In Figure 4.3 we can see a flow diagram for the system. In the next sections, each component will be explained in more detail.

4.3.1 Mujoco simulation controller package

This package abstracts the MuJoCo simulator, since its interface is not programmer-friendly. This API provides all the functionality that is needed for the algorithm.

4.3.2 Servers

All the packages that contain the simple_action_server atritube must contain a callback function that is called when a message is received. This function is used to parse the message and begin the simulation processing that is being requested.

4.3.2.1 Tube Grasp simulator action server

The tube grasp simulator server is in charge of receiving the trajectories created by the client and sending them to MuJoCo using the API provided by [Leão, 2020] and talked about in 4.3.1. This package has to guarantee that the selected wire edge reaches all the intermediate waypoints of the trajectory using linear interpolation. This package accepts messages that respect the structure in listing Listing 4.2.

```
#goal definition
string worldFilename
Trajectory[] trajectories
tube_models_msgs/Tube[] wires
---
#result definition
string outcome
TrajectoryResult[] results
tube_models_msgs/Tube[] wires
---
#feedback
```

Listing 4.2: action file of tube_Grasp_Simulator_msgs package

Since the client does not to update any values when the trajectory is being executed, the feedback part of the message is not necessary. This server is also in charge of updating the values of the simulation using the package Mujoco Simulation Controller.

4.3.2.2 Weld action server

The weld action server is a simple server that receives a message, seen in Listing 4.3, with the bodies that the client is trying to weld. This weld connects 2 bodies together and makes it so that they do not separate. It must be specified in the MuJoCo XML, before the simulation has started, the bodies and state of the weld.

Once the server receives this message it checks if these bodies are within weld conditions (if the distance between then is very small) and if so welds them using the Mujoco Simulation

Controller package. This message does not need to return the result and the feedback since the server will be in an active loop checking the weld conditions until the simulation ends.

```
#goal definition
string[] bodies1
string[] bodies2
---
#result definition

---
#feedback
```

Listing 4.3: action file of tube_Grasp_Simulator_msgs package

4.3.3 Clients

The client side of the architecture is relatively more simple. To connect to the server, the client only needs the server name and the type of message that the server receives. Once this is met, the client can build a request message and send it.

4.3.3.1 Assembly Jig client

This package begins by creating the model for the simulation and writing to a file so the package referred in the subsubsection 4.3.2.1 can begin the MuJoCo environment.

Once the models are built, the client sends a message (Listing 4.3) to the weld action server (subsubsection 4.3.2.2) with the wire ends and the connectors.

After the server had been notified, the client starts to calculate the points that the cable must pass in order to successfully assemble it in the jig. It does one fork at a time, with a connector in the beginning and one at the end, and waits for the server's response before calculating the next points. These points are passed to the Tube grasp simulator action client described in subsubsection 4.3.3.2, so the message can be built and sent to the Tube grasp simulator action server. When the trajectory is ends, the response contains the attitudes of the wires, enabling the client to update his wires and thus can calculate the trajectory to the next point.

4.3.3.2 Tube grasp simulator action client

This package only receives the components of the message, creates the message, sends it to the Tube grasp simulator action server, referred in subsubsection 4.3.2.1, and waits for the response.

4.4 Wire assembly algorithm

Once the modeling part of the system is done, the algorithm that assembles the cables can begin. The algorithm 1 together with Figure 4.3 provides the sequence of events that define this algorithm.

Algorithm 1 Overview of the wire assembly algorithm

Input:

wire - Flexible wire

jig - Assembly jig

1: $(firstCylinder, lastCylinder) \leftarrow wire.GetEndPointCylinders()$
2: $(firstConnector, secondConnector) \leftarrow jig.GetConnectors()$
3: $wire.MoveCylinderTowards(firstCylinder, firstConnector)$
4: $totalDistance \leftarrow 0$
5: $prevPoint \leftarrow firstConnector.GetUpperAuxiliaryPoint()$
6: **for each** $fork \in jig.GetForks()$ **do**
7: $(upperAuxiliaryPoint, lowerAuxiliaryPoint) \leftarrow fork.GetAuxiliaryPoints()$
8: $totalDistance \leftarrow totalDistance + EuclideanDist(prevPoint, upperAuxiliaryPoint)$
9: $cylinder \leftarrow wire.GetCylinderAtDistance(totalDistance)$
10: $wire.MoveCylinderTowards(cylinder, upperAuxiliaryPoint)$
11: $wire.MoveCylinderTowards(cylinder, lowerAuxiliaryPoint)$
12: $prevPoint \leftarrow upperAuxiliaryPoint$
13: $wire.MoveCylinderTowards(lastCylinder, secondConnector)$

First, the name of the bodies of the connector and the name of the ends of the cable are sent to the weld action server, so that when the cable is placed correctly the cables can be welded to the connector. This process can be seen in the Algorithm 1 in lines 1 and 2.

The system moves both ends of the wire into the connectors, using the *MoveCylinderTowards* of the Algorithm 1 with the first argument being the capsule the gripper must grasp and the second being the position, and the wire must pass through all the forks present in the assembly jig as seen in Figure 4.4. The whole process of assembly could be seen in Appendix A and Appendix B. The gripper's position at a given time can be determined by a linear interpolation of the points, since it has a fixed velocity and the time at which the trajectory started, it is possible to calculate the time at which the trajectory will end, and so update the gripper's position over time. This method also allows using a SLERP (spherical linear interpolation) function to change the gripper's orientation.

Since the forks have all the parameters described in 4.1.1, the problem becomes more of complex. Furthermore, the table is rotated 60 degrees and thus the gravity does not work on our favor.

To make the assembly possible, two auxiliary points must be determined for each fork: The exact point above the assembly point and a point that helps to guide the cable through the fork, seen in Figure 4.5 and in line 7 of the Algorithm 1. We can see how both points are calculated in the following equations. The first point, represented in Figure 4.5 by point A, is directly above the fork and functions as a starting point to the assembly. The second point is positioned 5 centimeters away and in the same direction as the fork, seen in Figure 5.2 as point B, and acts as a guide to the wire in order to place correctly inside the fork.

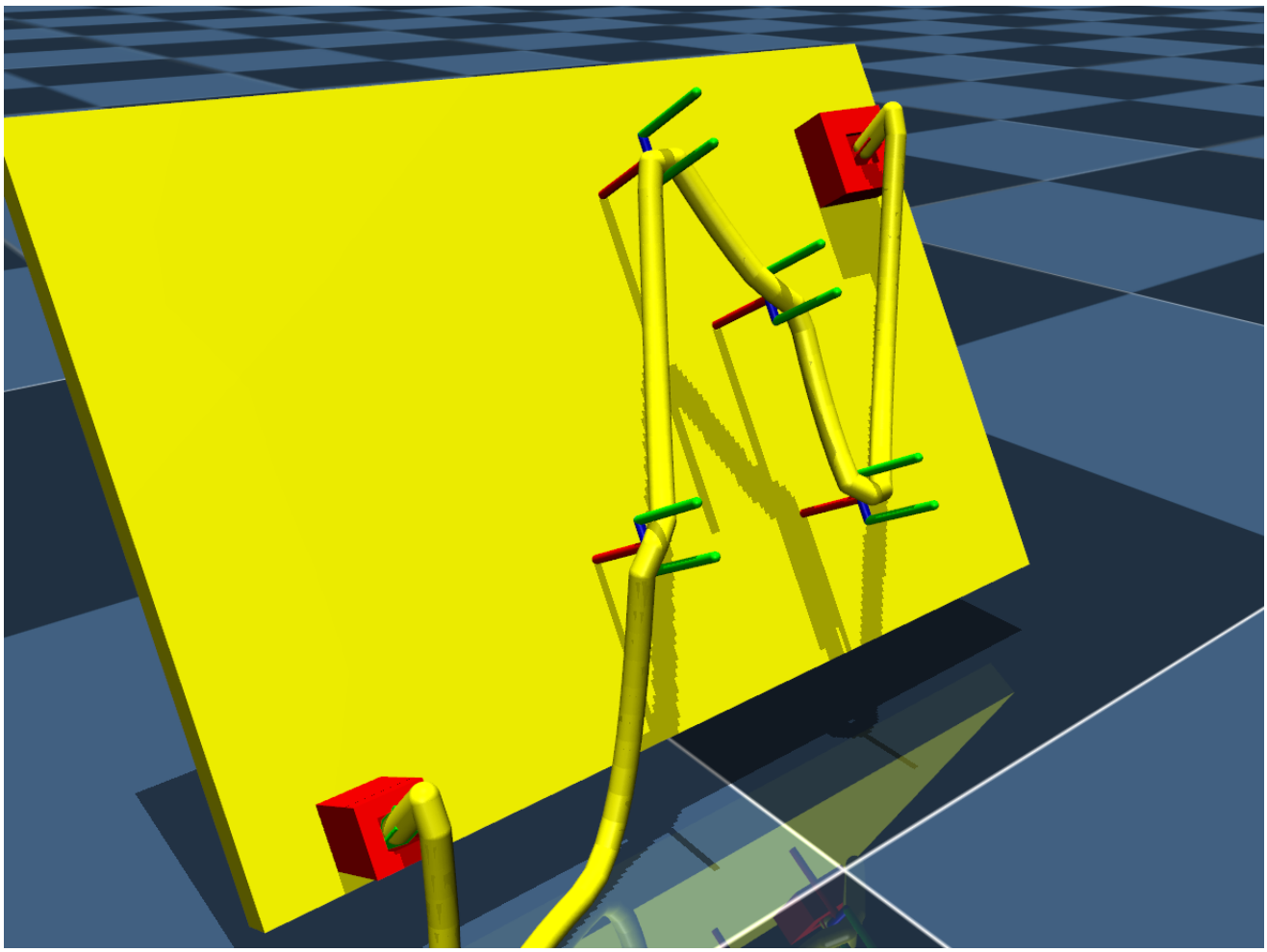


Figure 4.4: Example of a successful assembly

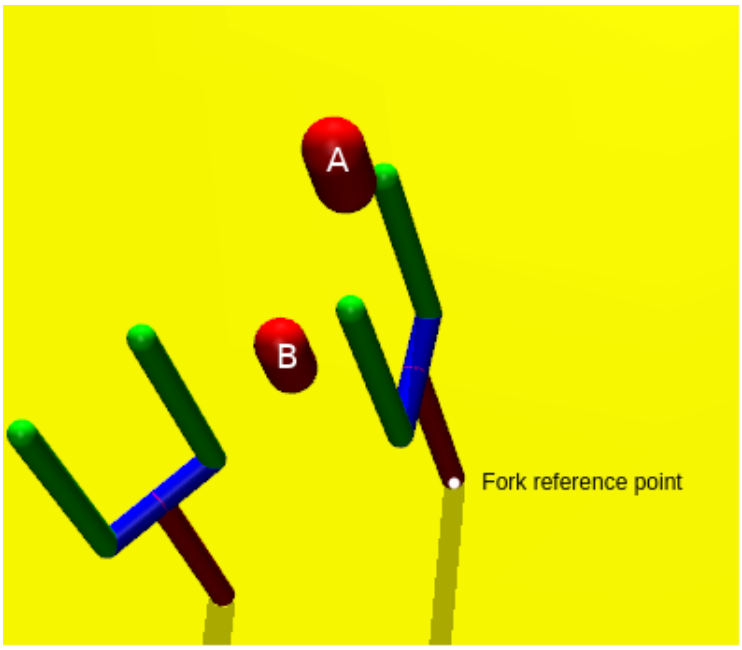


Figure 4.5: Reference points for the path of the cable. A-First point B-Second point

$$X_a = table.X - fork.X$$
$$Y_a = table.Y - (cos(table.angle) * fork.Y) + (cos(90 - table.angle) * fork.height)$$
$$Z_a = table.Z - (sin(table.angle) * -(fork.Y) + (sin(90 - table.angle) * fork.height)$$
$$X_b = table.X - (fork.X + (0.05 * cos(fork.angle)))$$
$$Y_b = table.Y - (cos(table.angle) * (fork.Y + 0.05 * sin(fork.angle))) + (cos(table.angle) * fork.height)$$
$$Z_b = table.Z - (sin(table.angle) * -(fork.Y + 0.05 * sin(fork.angle))) + (sin(table.angle) * fork.height)$$

When these points are found, the part of the wire that must be grasped needs to be determined. To resolve this problem, the distance between the already assembled forks are measured until the connector is reached. With this distance, the cylinder that will be sent to the server to be grasped is identified. This distance starts at 0, line 4 of the Algorithm 1, and is incremented with the distance of the previous point and the current one, seen in line 5, 8, 12. With this distance, we can resolve the capsule that must be grasped, using the *GetCylinderAtDistance* function seen in line 9 of Algorithm 1.

Once these two points and the cylinder to grasp are discovered, the trajectory message that is going to be sent to the tube grasp simulator action server 4.3.2.1 can be assembled. Once these messages reach the server, they are decoded and added to the list of trajectories that are going to be executed using the *MoveCylinderTowards* function that is in line 10 and 11 of the Algorithm 1. When the trajectory has finished, the server transforms the wire attributes into a message that is sent to the client as seen in Figure 4.3.

4.5 Summary

In this chapter, the usage and the architecture of the system that was developed in this dissertation was described. First, the basic usage of the modeling part of the system is described, followed by the structure of the packages and finally the assembly algorithm.

Chapter 5

Results of the simulated assembly of flexible wires

5.1 Overview

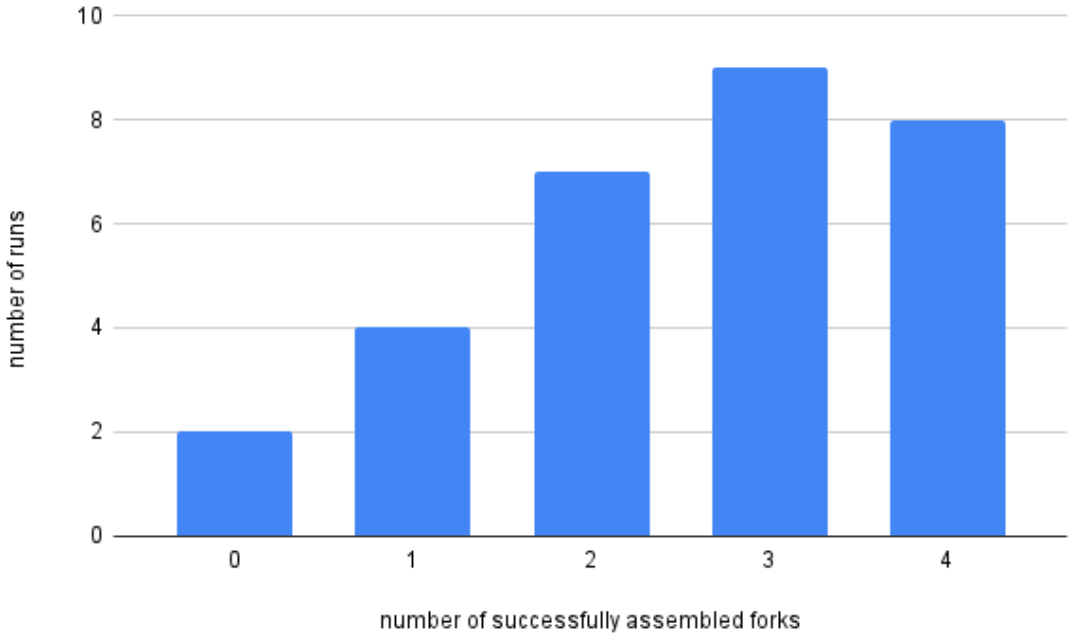
In order to measure the performance of the algorithm, the following metrics were measured:

- Number of forks that were successfully assembled;
- Time that took to conclude the assembly process.

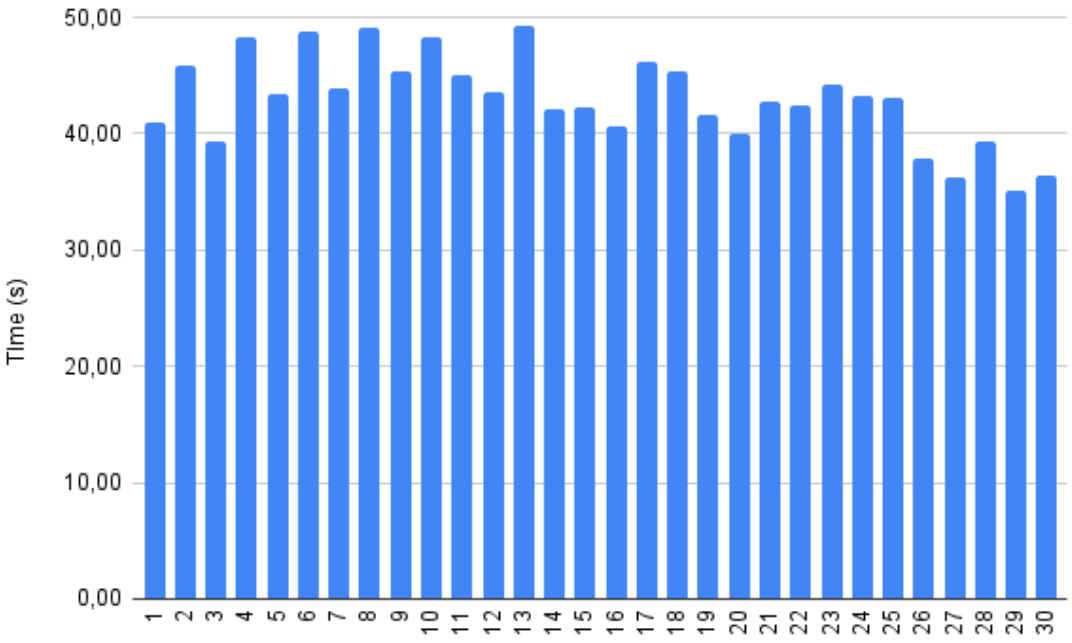
The number of forks that were successfully assembled, despite not giving a full picture of the assembly process, provide an easy way to measure the performance of the algorithm. The assembly time, on the other hand, serves the purpose of determining whether the algorithm got stuck. In order to properly test the algorithm, the position and rotation of the forks were randomized. Since in the real world, the forks would not be very close together, in these experiences the random forks must be at least 10 cm apart. The results of each experiment are the average of 30 runs.

In this configuration, the algorithm managed to successfully assemble on average 2.56 forks and took on average 90.99 seconds to finish the assembly, with a standard deviation of 6.49 seconds. It is also relevant to mention that the wire gripper moves at a fixed rate of 30 cm/s, and that does not change throughout the experiments. The following sections only change one parameter from this configuration, and thus these results are the baseline that the next configurations are going to be compared to. In the next sections, the following configurations will be tested:

- 7 forks – This configuration will serve to test a more real configuration.
- Forks without rotation – In order to test the last item further, a configuration where all the forks have no rotation of 0, thus meaning that all forks are vertical.
- Forks that are rotated towards the previous fork – During the process of creating the results for the baseline, a pattern appeared. When the rotation is in disagreement with the previous fork, the current fork tends to not be successfully assembled, since the gravity will work against the algorithm.



(a) Number of forks that were successfully assembled in each run



(b) Time of each run

Figure 5.1: Metric of 4 forks with random positions and angles

5.2 Increasing the amount of forks

In this section, the system was tested with 7 forks, in order to assess if the algorithm could scale to handle large amounts of forks. The results are very poor with this configuration, as seen in Figure 5.2. This is due to an increase in the number of forks leading to an increase in the likelihood of the wire becoming entangled with the environment, since each fork has random position and orientation. As a result, there are no runs with more than 5 forks where the algorithm managed to successfully assemble the wire.

This configuration had an average success rate of assembled forks of 2.67 with is higher than the baseline, but since that this scenario had more forks it is actually a downgrade in comparison. The experiment had an average time of 101.63 seconds and with a standard deviation of 1.52 seconds.

5.3 No rotation

In this configuration, there will be only 4 forks that do not have any rotation. If we compare this scenario with the baseline, we can determine if the rotation of the fork is of importance.

In Figure 5.3 it is possible to see the results of this configuration. The results were much better, with a fork success rate of 78.3% and the majority of runs managed to be fully assembled. With this experiment, it is possible to conclude with greater confidence that the rotation is the main variable that changes the results. The experiment had an average time of 75.73 seconds and with a standard deviation of 7.47 seconds.

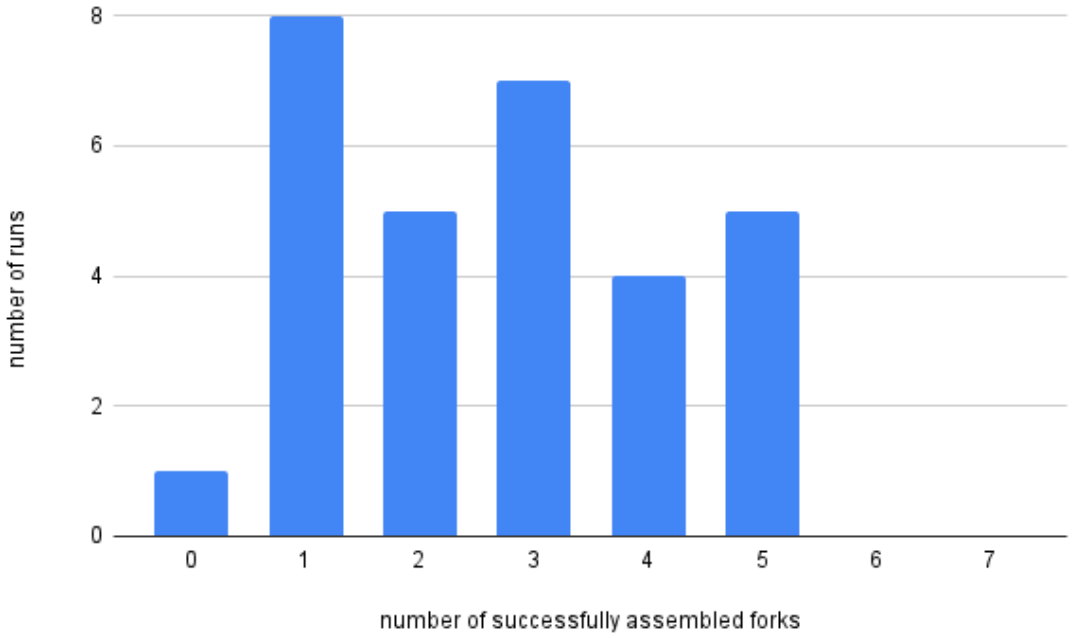
5.4 Rotation towards the previous fork

In order to further assess the effects of the rotation, in this experiment the forks are rotated towards the last assembled fork, as seen in Figure 5.4. This experiment tries to mimic the configuration of a real scenario. To get the angle between the forks, first the fork vector must be sorted from right to left, and the following equation can be applied.

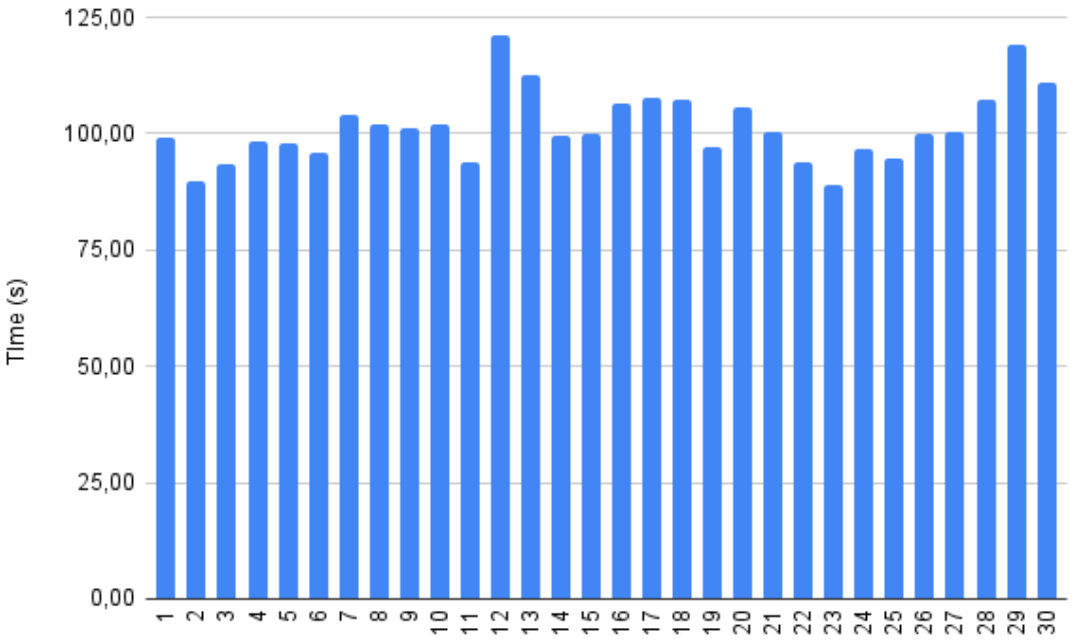
$$\theta = \text{atan}((Fork_i.y - Fork_{i-1}.y)/(Fork_i.x - Fork_{i-1}.x)) \tag{5.1}$$

With this equation, we can learn the angle that each fork must have in order to be in line with the previous fork. It is also important to refer that the first fork have no rotation.

This experiment had a fork success rate of 87.5%, that is the best result so far, despite the simplicity of the wire assemble algorithm. The experiment had an average time of 87.47 seconds and with a standard deviation of 6.27 seconds.

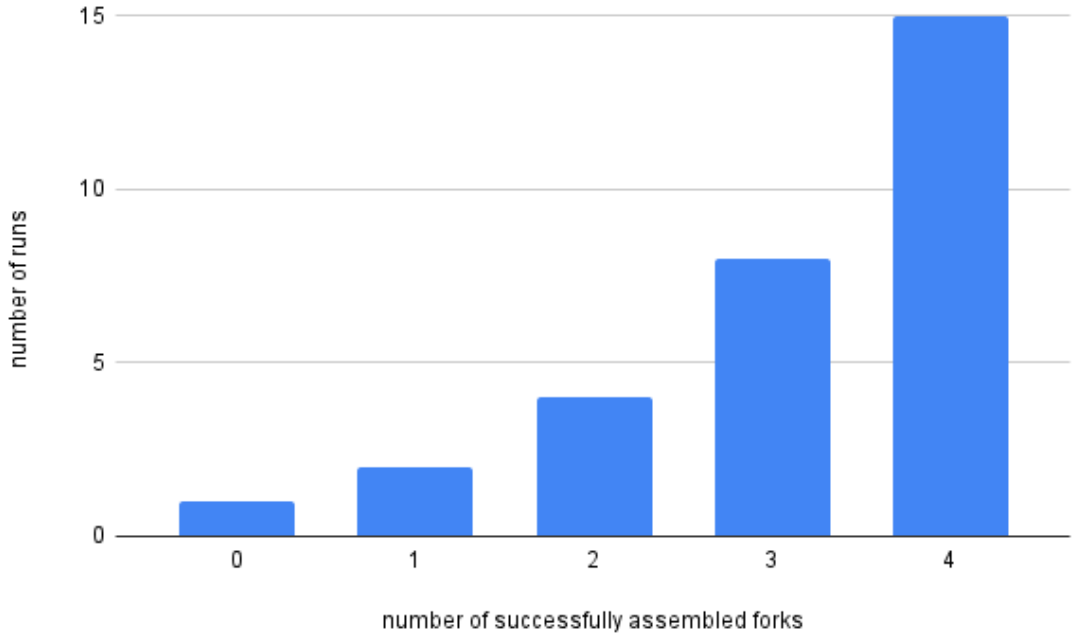


(a) Number of forks that were successfully assembled in each run

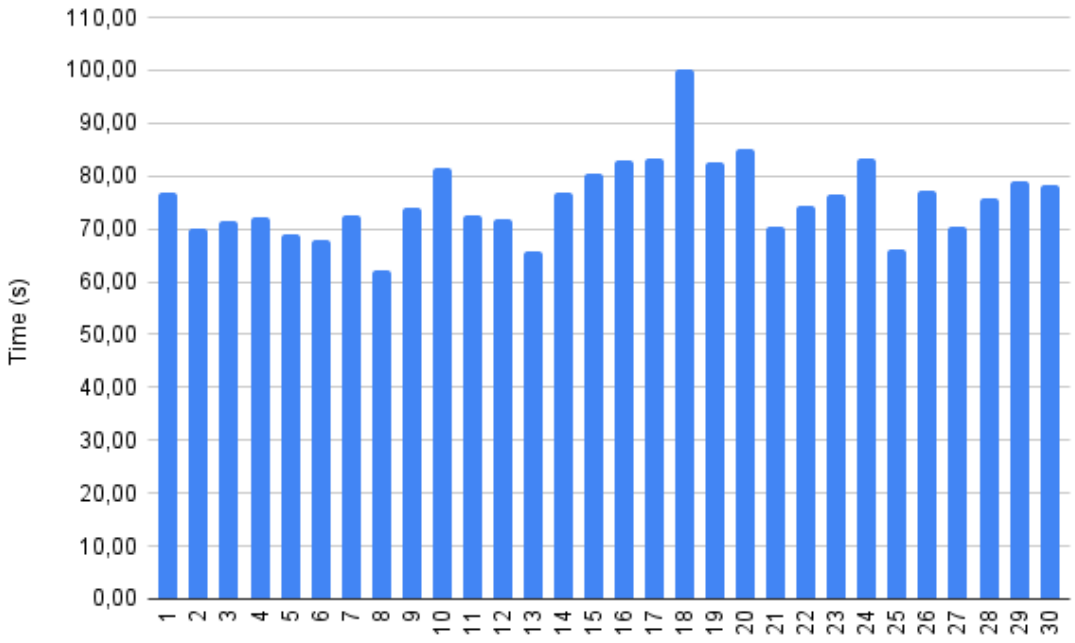


(b) Time of each run

Figure 5.2: Metric of 7 forks with random positions and angles



(a) Number of forks that were successfully assembled in each run



(b) Time of each run

Figure 5.3: Metrics of 4 forks with random positions but without rotation

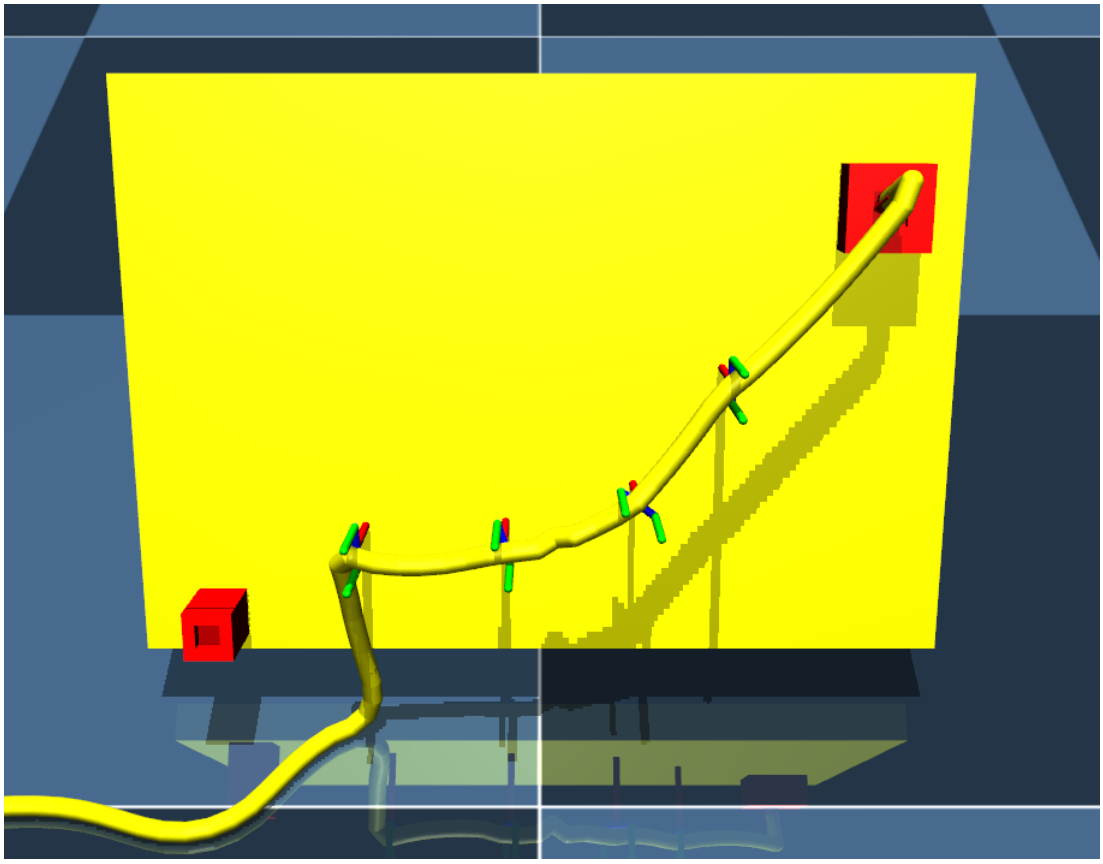
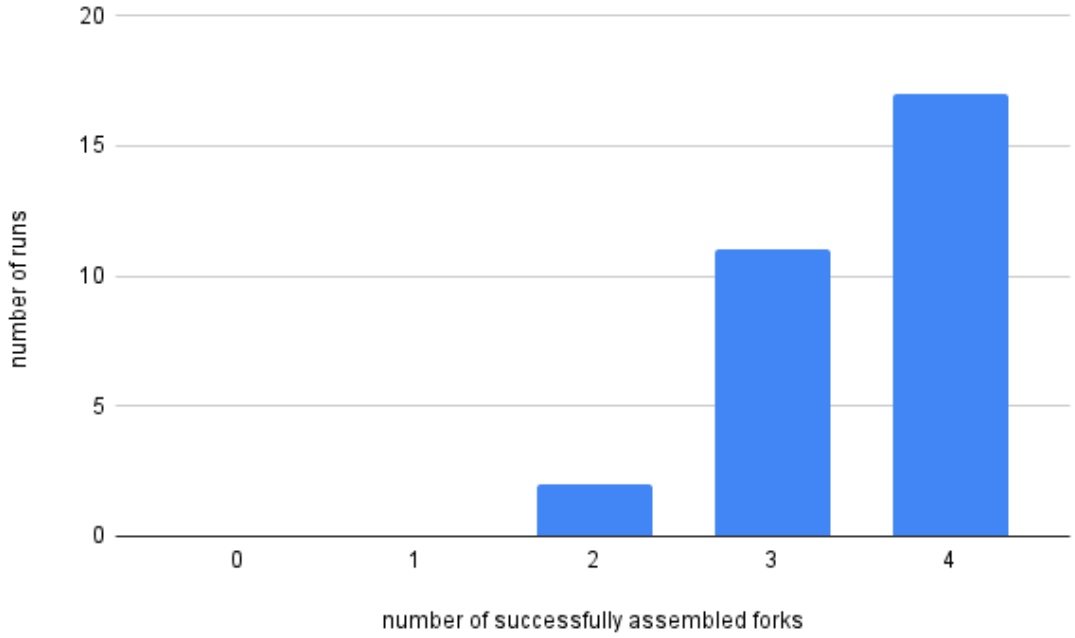
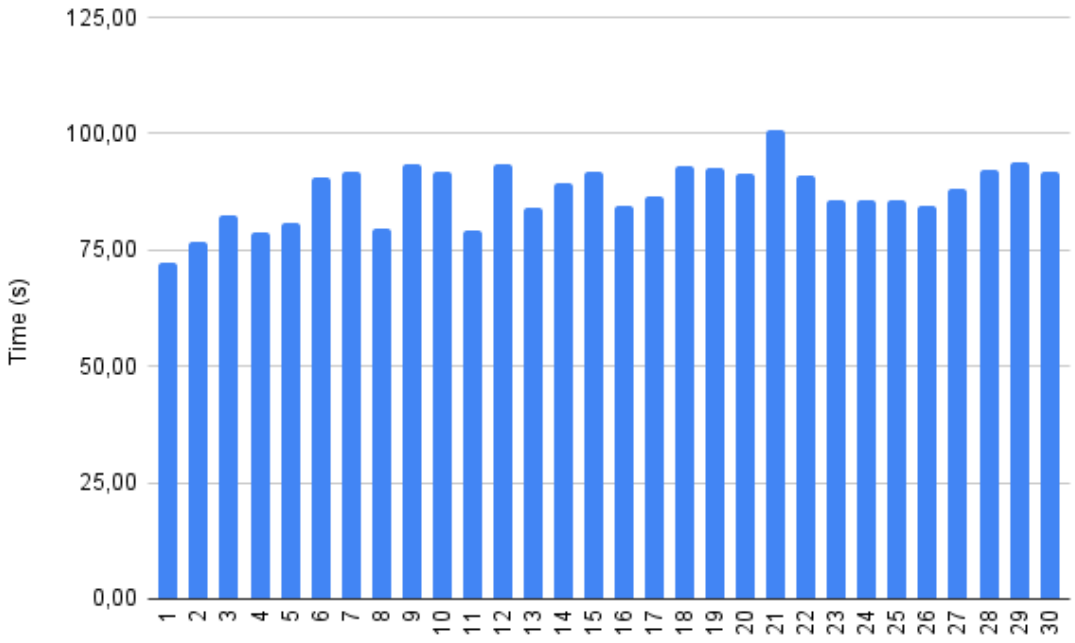


Figure 5.4: Fork configuration where the forks are rotated towards the last fork



(a) Number of forks that were successfully assembled in each run



(b) Time of each run

Figure 5.5: Metrics of 4 forks with random positions but rotation towards the last fork

5.5 Discussion

With the previous experiments, that each changed a parameter from the base configuration, we can conclude that the position of the forks and the rotation are more important than the number of forks. One of the major problems is the limitations of the MuJoCo physics engine, since there are some artifacts that may occur during the execution time, this can be seen in Appendix A.

The experiment with more forks had less performance due to the size of the jig being small, at 60 centimeters by 50 centimeters, and so the position of the forks are very near each other.

This conclusion can be further seen in the section 5.4, that got a significant better performance than the others experiments. The results seen here could be better if the position of the forks were less random. The algorithm struggled to assemble forks that are directly above or under the previous fork and such have a greater angle. An example of a successful assembly with this configuration can be found in Appendix B. The parameters of this experiment were inspired in real use cases and so there is a closeness to a real scenario.

Both Appendix A and Appendix B were produced with the code present in `https://github.com/daviddinis/VideoFrameCutter`.

5.6 Summary

In this chapter, the experiments that were performed to the algorithm were explained, and their results were discussed. It began by establishing a reference configuration in order to be able to compare the next experiments to something.

Chapter 6

Conclusions

In chapter 2, with the state-of-the-art, it was seen that the research of the manipulation on the flexible wires has been poorly explored and the solutions that were found did not explore the assembly of vehicle wire-harnesses. With this in mind, this dissertation attempts to provide a feasible solution to this problem.

First, the assembly jig and the wire must be modeled. Following the research question, *How can an assembly jig be modeled in simulation?*, it was established that the MuJoCo simulation environment was going to be used. The simulator was chosen due to allowing for an easy modeling of the assembly jig and the wire using the MuJoCo's XML language. Each of these models have several properties that are fed to the modeling algorithm using the ROS params.

Regarding the research question *What is an adequately simple strategy to mount a flexible wire in an assembly jig?*, the strategy developed in this dissertation consists of grasping the end point of the wire and inserting it into the connector. Then, the wire is passed through each fork in order. For each fork, there were two points that were calculated in order to facilitate the assembly of wire in the fork. The first point is directly above the fork, and the second is five centimeters in the direction of the fork and has the same height as the fork's root. After the forks, the strategy tries to conclude the solution by assembling the last connector in the same fashion as the first one.

The performance of the system was measured by the average time spent and the number of times that the cable passed in the fork at each run. To validate the work with a slightly more realistic setup, an experience where each fork is rotated toward the fork that is immediately at the right was made. This experiment provides the best results, with a fork success rate of 87.5%, despite the simplicity of the wire assemble algorithm. The experiment had an average time of 87.47 seconds and with a standard deviation of 6.27 seconds to pass the cable along four forks. It is also important to mention that the jig table has dimensions of 1 meter by 70 centimeters by 5 centimeters and the wire had a length of 2.5 meters.

6.1 Main contributions

The main contributions of this dissertation are:

- A modular architecture that can be extended to provide extra functionality, such as different clients that provide the system with different motion planning strategies.
- A basic algorithm that can assemble the wires in the jig. This algorithm supports forks with various heights, thicknesses, positions and rotations.

6.2 Future work

With the work done in this dissertation, it is possible to research further this topic.

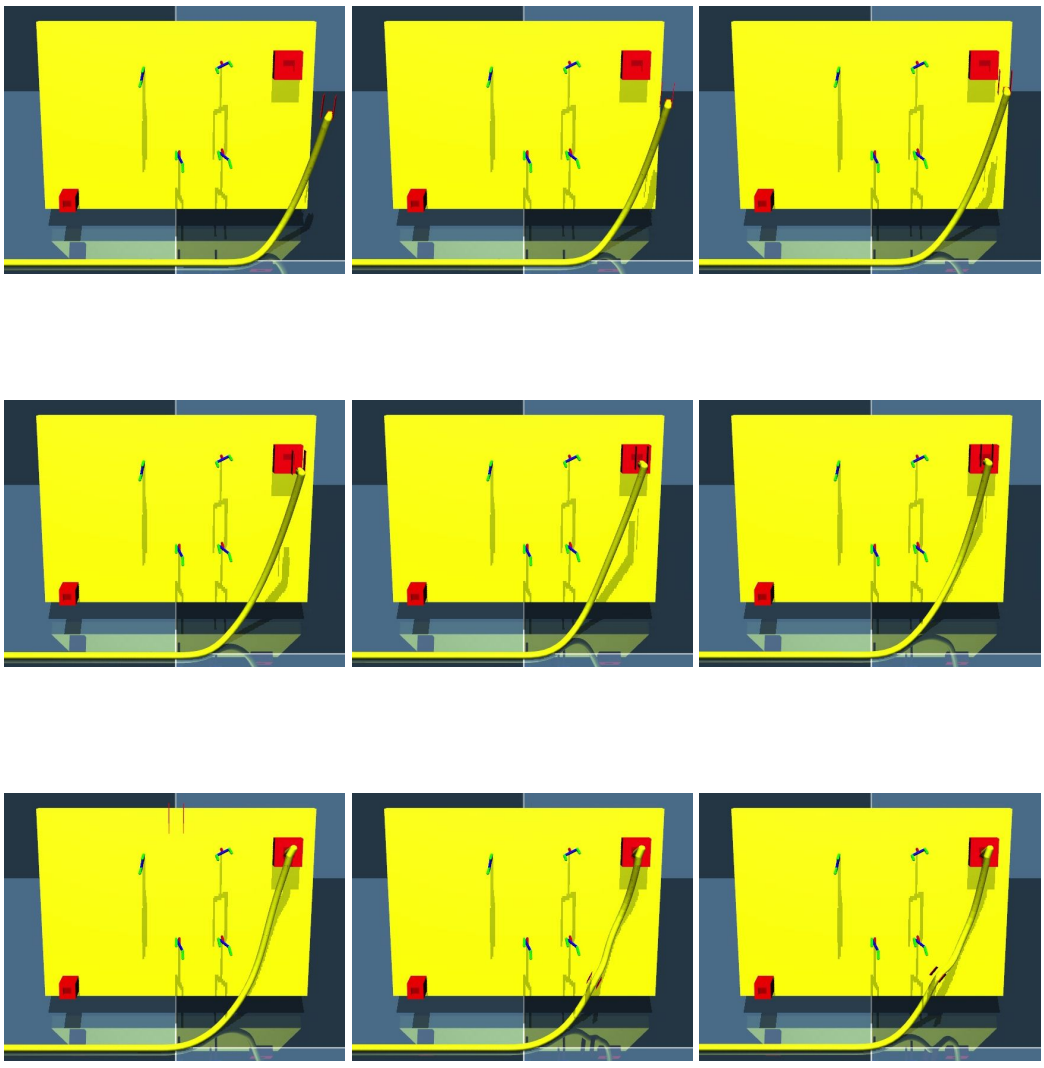
Since this solution is mainly developed for the automotive industry, it is important to reduce the execution time. This improvement will be restricted by the stability of the simulator, since if the simulation is sped up, there are cases of object clipping. With this in mind, it is important to refer that testing the system in real life is important to guarantee the robustness of the motion planning and the model system and to validate the solution that was implemented.

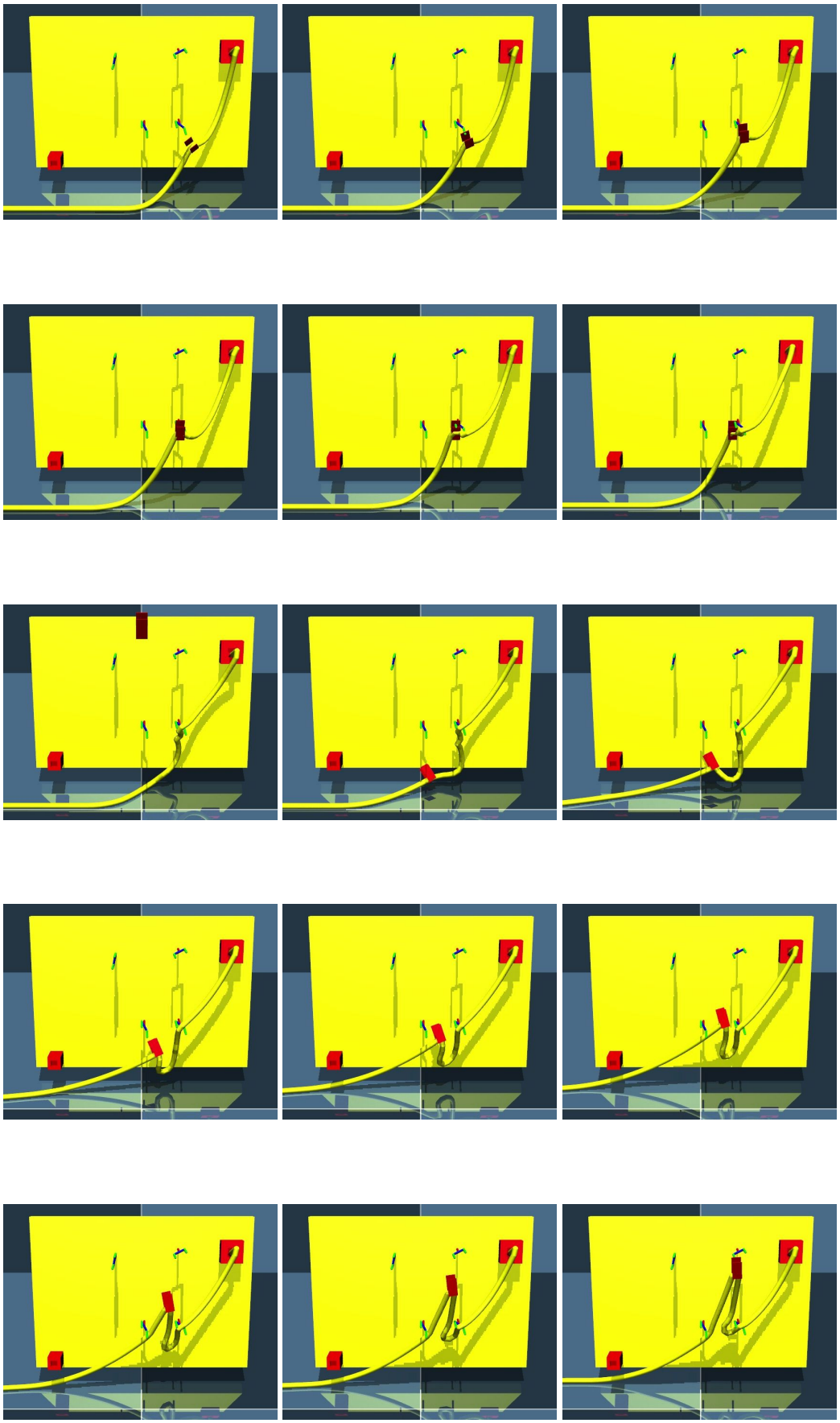
With the architecture approach presented in chapter 4, the next step to improve the solution can be to implement machine learning techniques in the motion planning phase. This can greatly improve the overall performance of the system by creating different auxiliary points to each fork, or the ability to change the velocity in different situations in order to provide advantages to the assembly process.

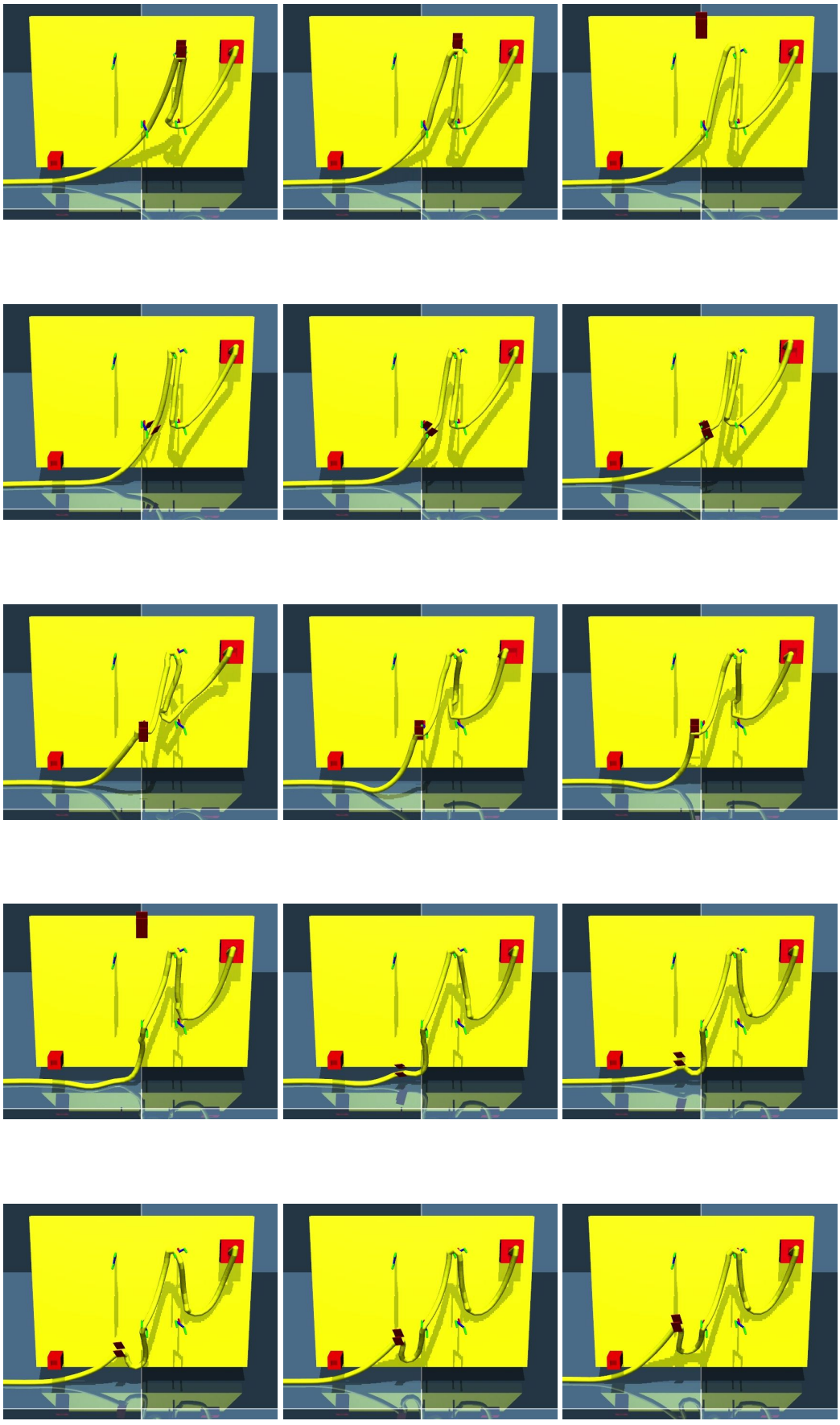
The current system only withstands one wire at a time, this could be a great opportunity to improve the algorithm if it could support various wires simultaneously.

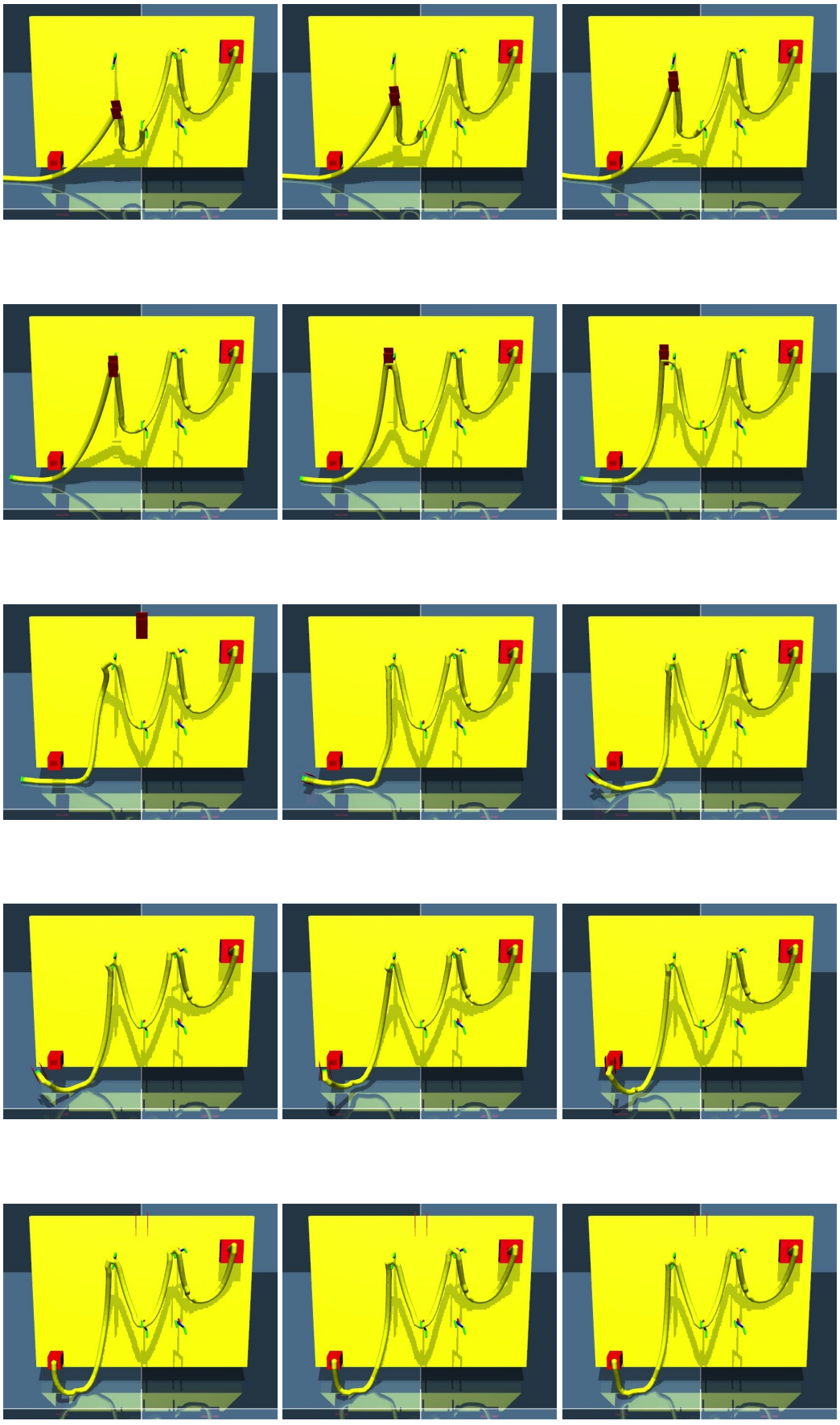
Appendix A

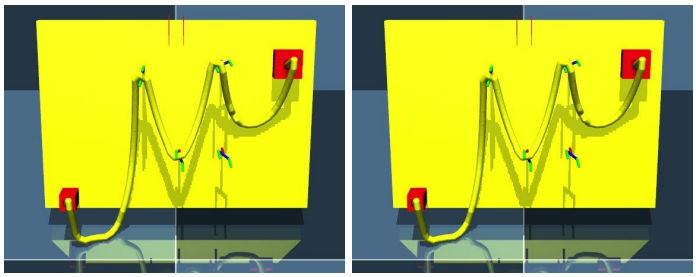
Example of assembly with random positions and random rotations





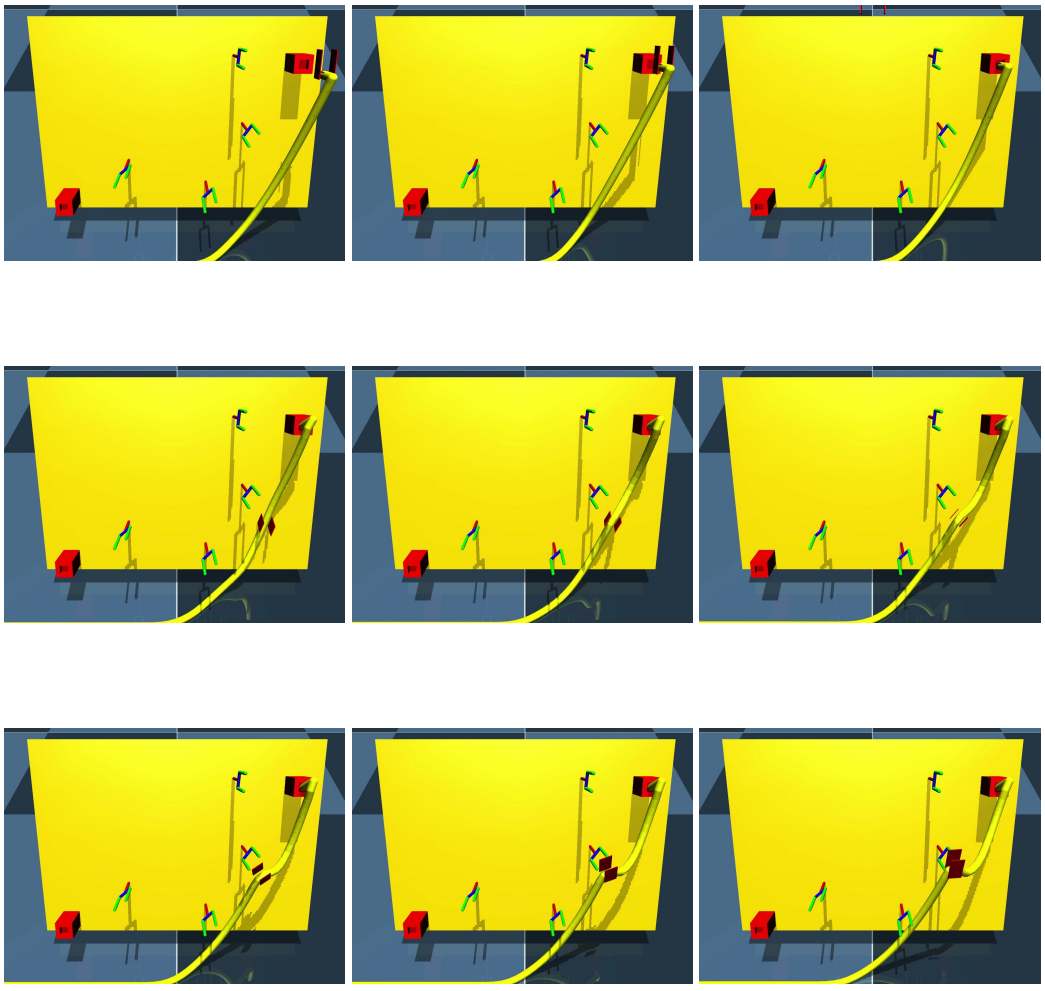


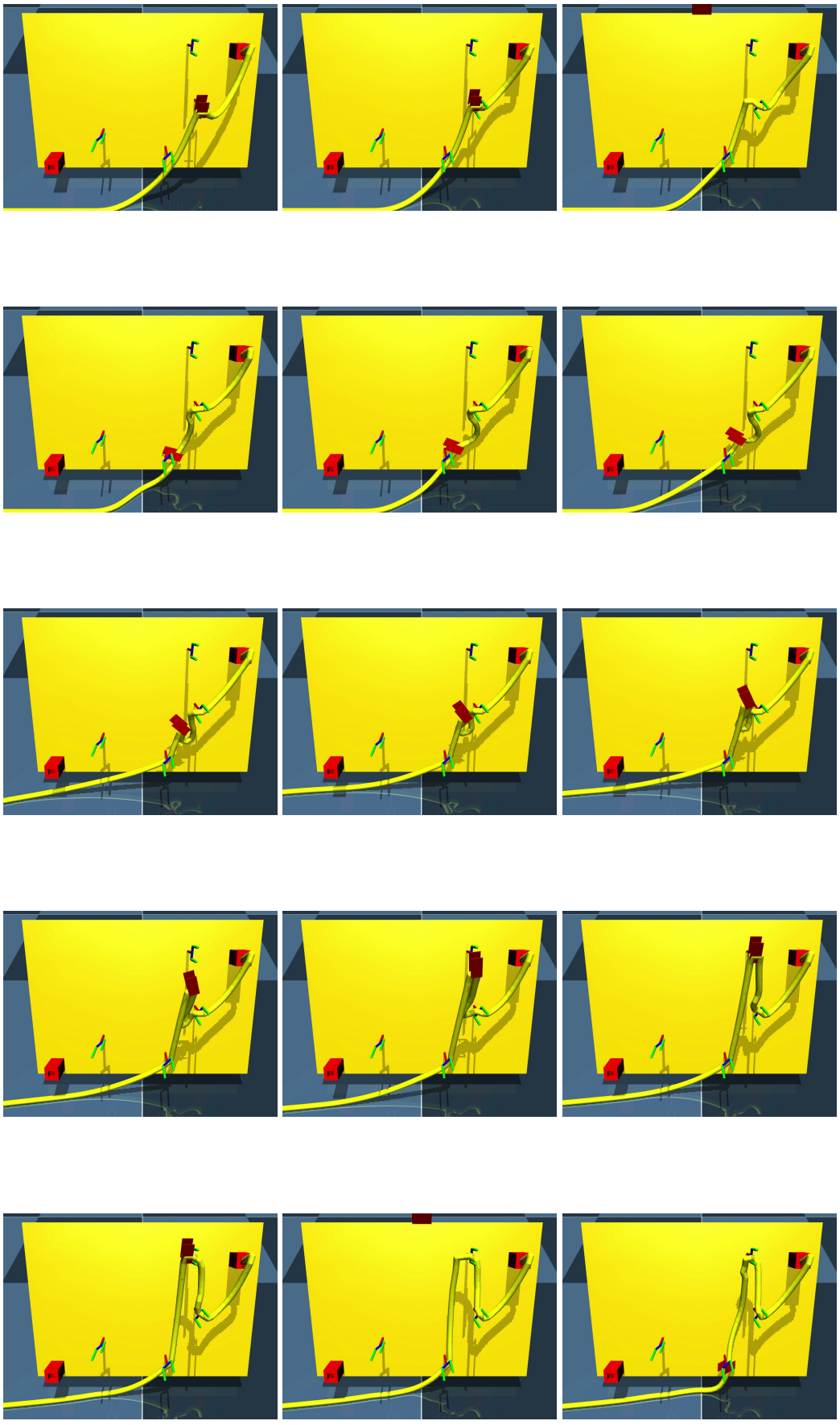


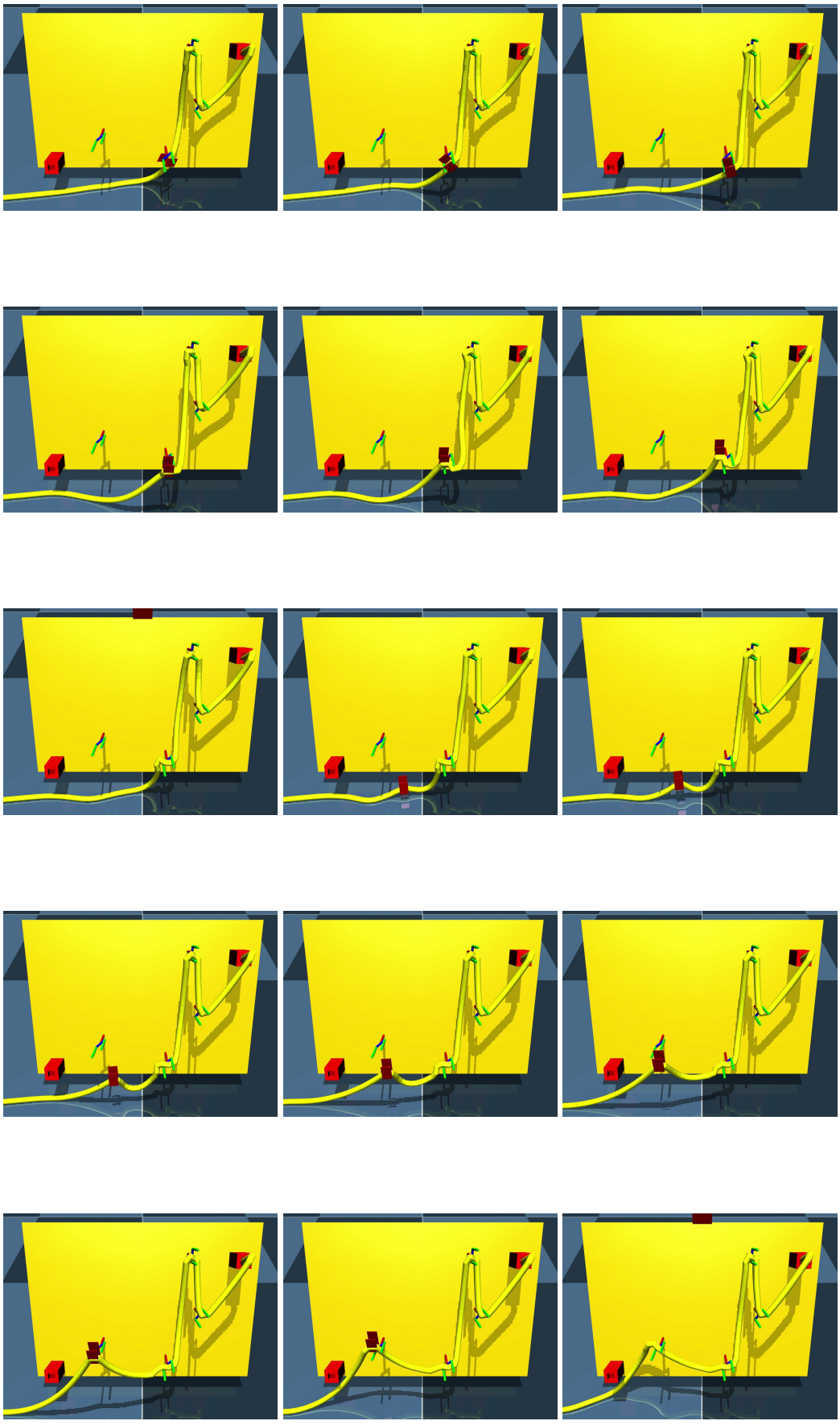


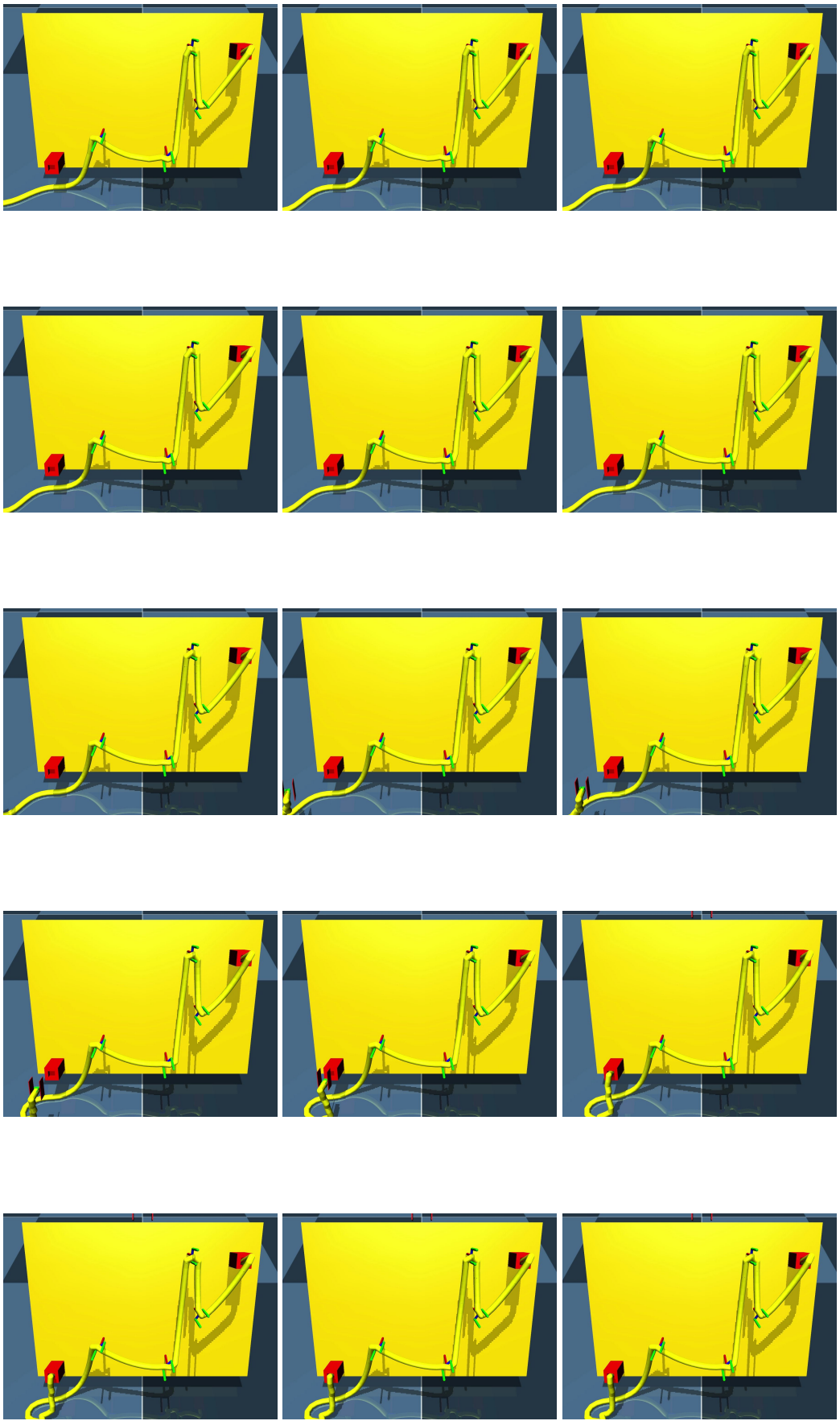
Appendix B

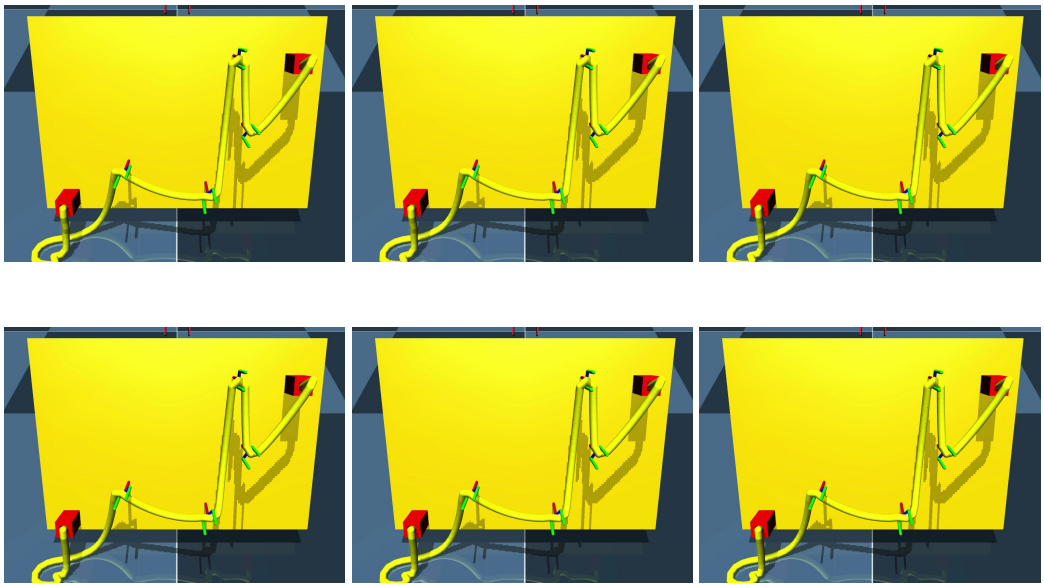
Example of assembly with random positions and rotated toward the previous fork











Bibliography

[Buchanan, 2005] Buchanan, B. G. (2005). A (very) brief history of artificial intelligence. *AI Magazine*, 26(4):53.

[Caporali et al., 2021] Caporali, A., Galassi, K., Laudante, G., Palli, G., and Pirozzi, S. (2021). Combining vision and tactile data for cable grasping. In *2021 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. IEEE.

[Jiang et al., 2011] Jiang, X., Koo, K.-M., Kikuchi, K., Konno, A., and Uchiyama, M. (2011). Robotized assembly of a wire harness in a car production line. *Advanced Robotics*, 25(3-4):473–489.

[Jiang et al., 2015] Jiang, X., Nagaoka, Y., Ishii, K., Abiko, S., Tsujita, T., and Uchiyama, M. (2015). Robotized recognition of a wire harness utilizing tracing operation. *Robotics and Computer-Integrated Manufacturing*, 34:52–61.

[Lamiraux and Kavraki, 2001] Lamiraux, F. and Kavraki, L. E. (2001). Planning paths for elastic objects under manipulation constraints. *The International Journal of Robotics Research*, 20(3):188–208.

[Leão et al., 2022] Leão, G., Costa, C. M., Sousa, A., Reis, L. P., and Veiga, G. (2022). Using simulation to evaluate a tube perception algorithm for bin picking. *Robotics*, 11(2):46.

[Lee et al., 2022] Lee, R., Hamaya, M., Murooka, T., Ijiri, Y., and Corke, P. (2022). Sample-efficient learning of deformable linear object manipulation in the real world through self-supervision. *IEEE Robotics and Automation Letters*, 7(1):573–580.

[Leão, 2020] Leão, G. (2020). Robotic bin picking of entangled tubes. Master's thesis.

[Leão et al., 2020] Leão, G., Costa, C., Sousa, A., and Veiga, G. (2020). Detecting and solving tube entanglement in bin picking operations. *Applied Sciences*, 10:2264.

[Lv et al., 2022] Lv, N., Liu, J., and Jia, Y. (2022). Dynamic modeling and control of deformable linear objects for single-arm and dual-arm robot manipulations. *IEEE Transactions on Robotics*, pages 1–13.

[Mahler et al., 2017] Mahler, J., Liang, J., Niyaz, S., Laskey, M., Doan, R., Liu, X., Ojea, J. A., and Goldberg, K. (2017). Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics.

[mo Koo et al., 2008] mo Koo, K., Jiang, X., Kikuchi, K., Konno, A., and Uchiyama, M. (2008). Development of a robot car wiring system. In *2008 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. IEEE.

[Moll and Kavraki, 2004] Moll, M. and Kavraki, L. (2004). Path planning for minimal energy curves of constant length. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*. IEEE.

[Moll and Kavraki, 2006] Moll, M. and Kavraki, L. (2006). Path planning for deformable linear objects. *IEEE Transactions on Robotics*, 22(4):625–636.

[Moravec, 2021] Moravec, H. P. (2021). robot. *Encyclopedia Britannica*.

[Papacharalampopoulos et al., 2015] Papacharalampopoulos, A., Makris, S., Bitzios, A., and Chryssolouris, G. (2015). Prediction of cabling shape during robotic manipulation. *The International Journal of Advanced Manufacturing Technology*, 82(1-4):123–132.

[Roussel and Taïx, 2014] Roussel, O. and Taïx, M. (2014). Deformable linear object manipulation planning with contacts. In *Robot Manipulation: What has been achieved and what remains to be done? Full day workshop at IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Chicago, United States.

[Saha and Isto, 2007] Saha, M. and Isto, P. (2007). Manipulation planning for deformable linear objects. *IEEE Transactions on Robotics*, 23(6):1141–1150.

[Sanchez et al., 2018] Sanchez, J., Corrales, J.-A., Bouzgarrou, B.-C., and Mezouar, Y. (2018). Robotic manipulation and sensing of deformable objects in domestic and industrial applications: a survey. *The International Journal of Robotics Research*, 37(7):688–716.

[Shah and Shah, 2016] Shah, A. J. and Shah, J. A. (2016). Towards manipulation planning for multiple interlinked deformable linear objects. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.

[She et al., 2019] She, Y., Wang, S., Dong, S., Sunil, N., Rodriguez, A., and Adelson, E. (2019). Cable manipulation with a tactile-reactive gripper.

[Shi et al., 2012] Shi, J., Hamner, B., Simmons, R., and Singh, S. (2012). Mobile robotic assembly on a moving vehicle. In *ASME/ISCIE 2012 International Symposium on Flexible Automation*. American Society of Mechanical Engineers.

[Veltkamp and Wesselink, 1995] Veltkamp, R. C. and Wesselink, W. (1995). Modeling 3d curves of minimal energy. *Computer Graphics Forum*, 14(3):97–110.

[Wu et al., 2019] Wu, Y., Yan, W., Kurutach, T., Pinto, L., and Abbeel, P. (2019). Learning to manipulate deformable objects without demonstrations.

[Zhu et al., 2018] Zhu, J., Navarro, B., Fraisse, P., Crosnier, A., and Cherubini, A. (2018). Dual-arm robotic manipulation of flexible cables. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.