|  |  |  |
| --- | --- | --- |
| **images** | **УНИВЕРЗИТЕТ “Св. КИРИЛ И МЕТОДИЈ” - СКОПЈЕ**  **ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА И ИНФОРМАЦИСКИ ТЕХНОЛОГИИ** | **index** |

- **ДИПЛОМСКА РАБОТА** -

по предметот

**РОБОТИКА 1**

**Тема**

**ФИЗИЧКА СИМУЛАЦИЈА НА РОБОТКСА РАКА**

|  |  |
| --- | --- |
| Ментор: | Изработил: |
| Доц. д-р Горјан Наџински | Васил Трендафилов, индекс бр. 139/2018  e-mail: vasetrendafilov@gmail.com |

*Скопје, декември 2022*

**Содржина**

[Апстракт 3](#_Toc121606742)

[Вовед 4](#_Toc121606746)

[1 Користени технологии и алатки 4](#_Toc121606747)

[2 Инсталација на софтверскиот пакет 4](#_Toc121606748)

[2.1 Инсталација на потребни апликации 5](#_Toc121606749)

[2.2 Инсталација на библиотеки и клонирање на проектот 5](#_Toc121606750)

[3 Користење на софтверскиот пакет 5](#_Toc121606751)

[3.1 Модуларна роботкса рака 5](#_Toc121606752)

[3.1.1 Креирање и пресметување на DH матрица и јакобијан на раката 6](#_Toc121606753)

[3.1.2 Инверзна кинематика на роботска рака со присматичен грипер 7](#_Toc121606754)

[3.1.3 Директна кинематика на роботска рака со револутен грипер 8](#_Toc121606755)

[3.1.4 Импортирање на рака и репродуцирање на логовите 9](#_Toc121606756)

[3.1.5 Пример за детекција и зимање на објект со грипер 10](#_Toc121606757)

[3.2 Популарни роботски раце 12](#_Toc121606758)

[3.2.1 Franka Emirka panda роботска рака 12](#_Toc121606759)

[3.2.2 xArm роботкса рака 13](#_Toc121606760)

[3.3 Mitsubishi роботска рака 14](#_Toc121606761)

[3.3.1 Контрола на Mitsubishi роботската рака со гриппер 14](#_Toc121606762)

[3.3.2 Поднесување на параметри на раката за контрола преку комадни 15](#_Toc121606763)

[3.4 Кратки копчиња за pybullet 17](#_Toc121606764)

[4 Надградување на софтверскиот пакет 18](#_Toc121606765)

[4.1 Додавање на нова алатка 18](#_Toc121606766)

[4.2 Додавање на новa роботска рака 18](#_Toc121606767)

[4.3 Креирање на нов свет во pybullet 18](#_Toc121606768)

[4.4 Нови функционалности на софтверскиот пакет 18](#_Toc121606769)

[4.5 Нови примери користејки го софтверскиот пакет 19](#_Toc121606770)

[5 Конектирање на CR750-Q со RT Toolbox 2 19](#_Toc121606771)

[6 Архитектура на системот 21](#_Toc121606772)

[6.1 Хиерархија помеѓу класите 21](#_Toc121606773)

[6.2 Хиерархија на класата PybulletSimulation 22](#_Toc121606774)

[6.3 Хиерархија на класата Camera 22](#_Toc121606775)

[6.4 Хиерархија на класата DH2Urdf 23](#_Toc121606776)

[6.5 Хиерархија на функцијата interact 23](#_Toc121606777)

[6.6 Хиерархија на контрола преку команди 24](#_Toc121606778)

[6.7 Хиерархија за креирање на модуларна роботска рака 24](#_Toc121606779)

[6.8 Модули за симулација на роботската рака 25](#_Toc121606780)

[7 Докс 25](#_Toc121606781)

[8 Заклучок 25](#_Toc121606782)

[9 Референци 25](#_Toc121606783)

**Листа на слики**

[Слика 1 дх матрица 6](#_Toc121595434)

**Листа на табели**

[Табела 1 Карактеристики на батерии кои може да се полнат 6](#_Toc222636361)

# Апстракт

*Автономните возила сега се неопходност во овој свет, што го прави апсолутно важно да се има звучен алгоритамски систем кој ќе врши откривање на различни карактеристики на патот, како што се маркери за лента, откривање возила итн. Во овој проект, лентите ќе бидат откриени со користење на canny и hough lines и алгоритам за филтрирање на трите ленти. Ќе ја имплементираме и невронската мрежа yolov5 за откривање возила на патот. Овој пајплајн во реално време ќе се имплементира на duckiebot за тестирање на перформансите на алгоритмот.*

# Клучни зборови: компјутерски вид, детекција на лента, автономно возење, yolov5

**Abstract**

# *Self driving cars are now a necessity in this world, which makes it absolutely important to have a sound algorithmic system that would perform detection of various features on the road, such as lane markers, detecting vehicles etc. In this project, lanes will be detected using canny and houg lines and an algorithm to filter out the three lanes. We will also implement the neural network yolov5 for detecting vehicles on the road. This real-time pipeline will be implemented on the duckiebot to test the performance of the algorithm.*

# Keywords: computer vision, lane detection, autonomous driving, yolov5

# Вовед

Главната цел на овој проект е да се создаде симулатор на роботска рака каде што можете лесно да се креира или вчита постоечки urdf и да управива со роботот во симулиран свет со генерирани пречки и предизвици. Симулаторот што се користи е pybullet кој доаѓа со сите потребни компоненти за изградба на обемен и комплициран симулиран свет. Основата за создавање на роботската рака е направена со sympy и сите алатки се извлечени од основите на роботиката. Исто така, постојат посветени пакети за градење на светот околу роботската рака и конвертирање на параметрите на Denavit–Hartenberg DH во urdf за да може да се увезат во симулацијата. Во делот robot\_arms/ има вистински раце како панда и ксарм. Исто така, тука е и раката Mitsubishi RV-2F-Q каде што програмата исто така може да ги конвертира дневниците во програмска датотека RT Toolbox и да ја изврши програмата на вистинската рака.

# Користени технологии и алатки

Софтверскиот пакет содржи три мегусебно поврзани библиотеки. Главната библиотека е pybullet кој е брз и лесен за употреба пакет за симулација на роботи и машинско учење, со фокус на трансфер од симулациијата во реланоста. Oбезбедува симулација на динамика нанапред, пресметување на инверзна динамика, напредна и инверзна кинематика, детекција на судир и пресметка за пресек на зраци. Воглавно во проектот се користи за симулирање на раката, креирање на светот околу неа и логирање на сите зглобови.

Основата за создавање на роботската рака е направена со sympy која е библиотека на пајтон за симболична математика. Се користи за пресметка на DH матрицата и јакобијанот на раката. Третата бибилиотека е numpy која се користи за олеснување на операциите со матрици и низи во прокетот.

# Инсталација на софтверскиот пакет

## Инсталација на потребни апликации

Основни софтверски апликации кои ги употребува пакетот се пајтон со соодветен инсталер на бибилиотеки и веб пребарувач за користење на јупитер тетратките. Популарна алатка за инсталирање на бибилотеките која се користи е мини конда кој е бесплатен минимален инсталатер за конда. Чекорите за инсталација на апликацијата може да се следат во следниот линк од нивната документација [1].

Освен веб пребарувач, јупитер тетратките може да се отворат и со софтверскиот пакет Vscode каде имаме вградена екстензија за нивно вчитување. Како последен софтверски пакет кој е опционален е гит кој е бесплатен и дистрибуиран систем за контрола на верзии со отворен код. Со кој може лесно да се клонира проектот и понатаму да се надградува.

## Инсталација на библиотеки и клонирање на проектот

Потребни библиотеки кои проектот ги користи се pybulllet, sympy и numpy. Верзијата на пајтон нема потреба да се одбира бидејки пакетите за инсталирање на библиотеките автоматски ја одбира заедничката верзија за сите библиотеки. Со следните линии команди ќе се покажат чекорите потребни за колинирање на софтверскиот пакет, инсталирање на потребните библиотеки и стартување на јупитер лаб серверот:

1. Клонирање на рaкетот:

git clone <https://github.com/vasetrendafilov/robot-simulation>

1. Креирање на околината и инсталирање на портребните библиотеки:

conda create --name robot\_simulation -c conda-forge pybullet jupyterlab sympy

1. Активација на околината

conda activate robot\_simulation

1. Стартување на јупитер лаб за отварање на јупитер тетратките

jupyter lab

# Користење на софтверскиот пакет

Овој проект доаѓа со три јупитер тетратки каде што се имплементирани сите функционалности на софтверскиот пакет. Во следните подточки ќе се објаснат сите јупитер тетратки во детали со сите чекори како да се креира или импортира роботкса рака, контрола со неа и објаснување на сите модули на симулацијата.

## Модуларна роботкса рака

Може да се креираат модуларни роботски раце користејќи (DH - Denavit–Hartenberg) конвенцијата. Примери за контрола со нив и пресметка на DH матрица на роботската рака и линиарен и ангуларен јакобијан на роботсктата рака.

На почеток се полнат библиотеките sympy и numpy кои ќе се користат подоле за креирање на симболички променливи и олеснување при работа со матириците. Следно се импортираат класите RobotArm за креирање на робтоска рака и PybulletSimulation за креирање на светот околу раката.

import numpy as np

import sympy as sp

from packages.robot\_arm import RobotArm

from packages.pybullet\_sim import PybulletSimulation

### Креирање и пресметување на DH матрица и јакобијан на раката

Со sympy се креираат симболички променливи за параметрите на зглобовите. Потоа се користи класата RobotArm() каде прво се инијализира и се додаваат зглобови според DH конвецијата. Освен четирите основни параметри (тета, d, а, алфа), во функциите дополнително може да се сетираат долни и горни граници, максималната брзина и забрзување на згобот. Ако се додадат дополнителни симболички променливи, треба да се додаде на нив и нумеричка вредност со функцијата add\_subs(). На крај се полни раката во симулацијата со функцијата load\_robot().

d1, theta2, theta3, d4, l2, l3 = sp.symbols('d1, theta2, theta3, d4, l2, l3')

robot = RobotArm()

robot.add\_prismatic\_joint(0, d1, 0, 0)

robot.add\_revolute\_joint(theta2, 0, l2, 0)

robot.add\_revolute\_joint(theta3, 0, l3, sp.pi)

robot.add\_prismatic\_joint(0, d4, 0, 0)

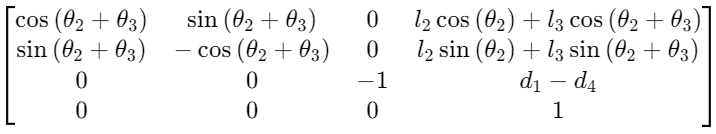
robot.add\_subs([(l2, 2), (l3, 2)])

robot.load\_robot()

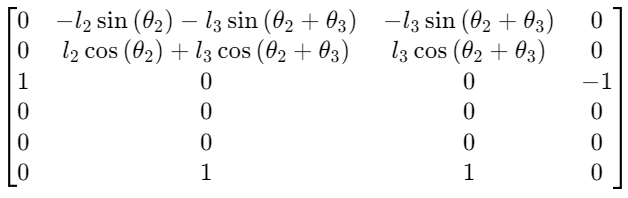
Со функцијата get\_dh\_matrix се пресметува дх матрцата за целата рака, со која се преставува зависноста на крајниот енд ефектор од позицијата на секој зглоб. Функцијата jacobian го пресметува линеарниот и англуарниот јакобијан за целата роботска рака.

robot.get\_dh\_**m**atrix()

robot.jacobian()



Слика дх матрица



### Инверзна кинематика на роботска рака со присматичен грипер

Вториот пример креира модуларна роботка рака на која се поставува присматичен гриппер и интерактивно се контролира раката. Се користат истите чекори за креирање на модуларната рака како првиот пример. За додавање на гриперот се косити функцијата add\_attachment која по дефаулт се додава присматичниот гриппер. За крај се повикува главната функција interact() каде се креира интерактивно гуи во пубуллет за контрола на раката со инверзна кинематика каде по дефаулт се користи вградената нумерицка апроксимација.

theta1, theta2, theta4, d1, d2, d3, a1, a2 = sp.symbols('theta1, theta2, theta4, d1, d2, d3, a1, a2')

robot = RobotArm()

robot.add\_revolute\_joint(theta1, d1, a1, 0)

robot.add\_revolute\_joint(theta2, 0, a2, -sp.pi)

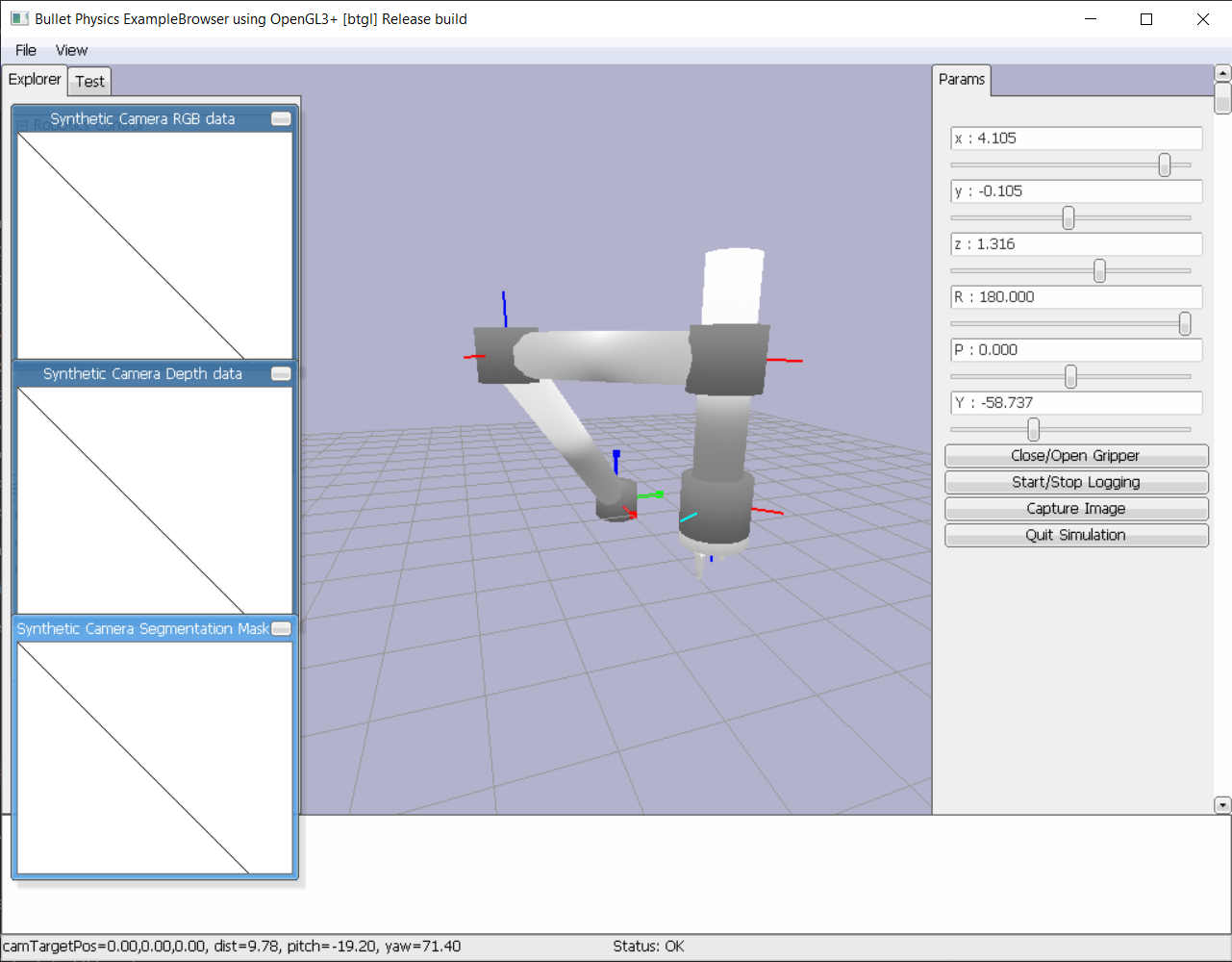
robot.add\_prismatic\_joint(0, d3, 0, 0)

robot.add\_revolute\_joint(theta4, 0, 0, 0)

robot.add\_subs([(d1, 3),(a1, 3),(a2, 3)])

robot.add\_attachment()

robot.interact('inverse')

Гуито на пѕбуллет се составува од главен прозорец каде се прикажува генерираниот свет, каде во моментов е само преставена рообтската рака. На десната страна се генерирани слајдери и копциња за контрола на раката. Кога се користи инверзна кинематика се поставуваат три слајдери за позиција на енд ефекторот по оските каде по дефаулт е од -5 до 5, и 3 слајдери за ориентацијата по конвенција RPY каде имаме ротации по x, y и z оски. Исто така имаме и копчиња за затворање и отворање на гриперот, да се запоцне и запре зачувањето на позициите на зглобовите, зимање на слика од енд ефекторот, и на крај исклучување на симулацијата.

Бројка 1

### Директна кинематика на роботска рака со револутен грипер

Следниот пример покажува контрола на модуларна роботска рака со револутен гриппер. Кога се додава нов додаток на раката треба да се постави оффсетот и ориентацијата од енд ефекторот и да се постават таргетите за точно колку да се отвора и затвора гриперот. Исто така имаме и случај при контрола на раката динамички имаме голема грешка во позицијата на зглобовите за ротарија. Тоа се дешава бидејки во примерот имаме многу зглобови кои ротираат во место и симулацијата неможе да го измоделира тоа. Има решениае и за тоа каде во римулацијата само ги ресетираме позициите на зглобовите и работи супер, но во вој мод неможе да се користи гриперот. На крај ја контролираме роботксата рака до директна кинематика кој е дефалтната опција.

theta1, d2, d3, theta4, theta5, theta6 = sp.symbols('theta1, d2, d3, theta4, theta5, theta6')

robot = RobotArm(use\_dynamics=True)

robot.add\_revolute\_joint(theta1, 0, 0, 0)

robot.add\_prismatic\_joint(0, d2, 0, -sp.pi/2)

robot.add\_prismatic\_joint(0, d3, 0, 0)

robot.add\_revolute\_joint(theta4, 0, 0, sp.pi/2)

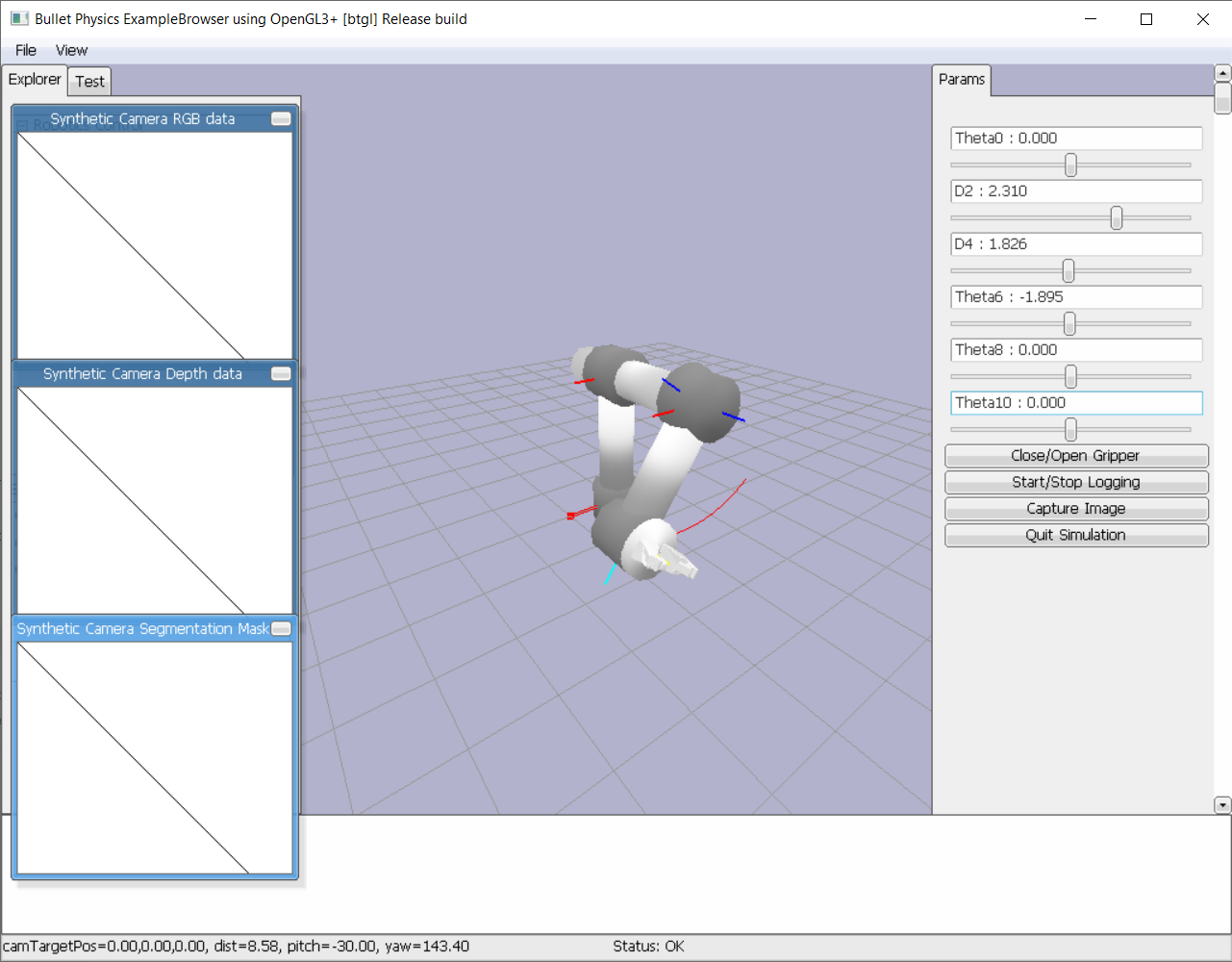
robot.add\_revolute\_joint(theta5, 0, 0, -sp.pi/2)

robot.add\_revolute\_joint(theta6, 0, 2, 0,)

robot.add\_attachment('revolute\_gripper',orientation = (0,-90,0))

robot.set\_attachment\_targets((0.548,0.548),(0,0))

robot.interact()

За контрола на роботската рака со директна кинематика имаме мали промени, каде сајдерите за промена на посизијата и ориентација на ендефекрот се заменети со слајдер за секој зглоб во раката кој е во поставените гранизи на секој зглоб. Исто така на сликата се прикажува и траекторијата на енд ефецторот каде во вој случај се гледа како се однесува но нишало пробувајќи да ги постави згобовите на точната позиција.

### Импортирање на рака и репродуцирање на логовите

Следниот пример се прикажува како да се импортира веќе креиран робот кој е направен од проектот и како да се реплеј направените логови од раката. При реплејнување на логовите имаме опција да миниме низ секој чекор секвенцијално или да ообереме преку слајдер кој чекор да го помине.

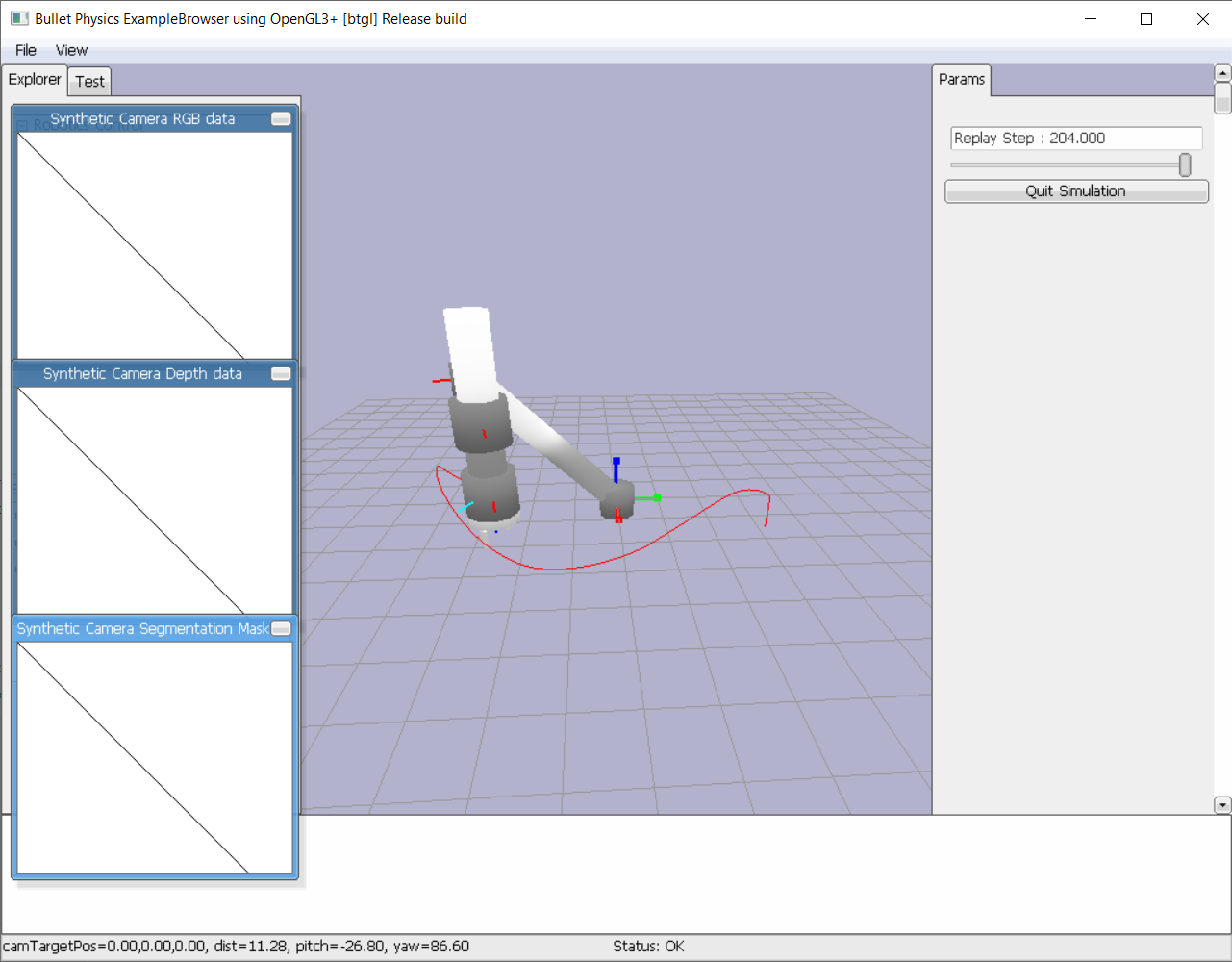
robot = RobotArm()

robot.import\_robot('robot\_arms/my\_robot/my\_robot.urdf')

robot.load\_robot()

robot.replay\_logs('inverse',skim\_trough=True)

Во сликата подоле е преставено гуито за реплејнување на секој чекор од логот.



### Пример за детекција и зимање на објект со грипер

Следниот пример е симпле пример за детекција на објект и зимање на објектот со грипперот на роботската рака. Прво се создава светот околу роботксата рака користејки ја каласата PybulletSimulation(), каде го стартуваме и конектираме со гуито на пѕбуллет. Го градиме светот така што поставуваме работна маса, и лего на случајан позиција на масата. Следно ја креираме роботската рака и ја поставуваме на едниот дел од масата. За да имаме прецизни двизења силата на зглобовите ја намалуваме на 400 и ги поставуваме динамичките цондиции да грешката во згобовите да е 10^-3 и да чека 1000 степс за да запре ако не успее да стигне до саканата позиција.

sim = PybulletSimulation()

sim.connect()

sim.load\_table()

# place a lego randomly on the table

sim.load\_lego((np.random.uniform(-1.5,1.5), np.random.uniform(-1.5,1.5),1),scaling=10)

robot = RobotArm((-3,0,0.4),joint\_forces=400)

robot.import\_robot('robot\_arms/my\_robot/my\_robot.urdf')

# set the dynamic condition to have grater precision

robot.set\_dynamic\_conditions(1000,0.001)

robot.load\_robot()

Со следниот блок код се објаснува како работи краткиот алгоритам да го премознае објектот и да го фати. Прво енд ефекторот на роботската рака се поставува на центарот на масата и на највисоката позиција која може раката да ја постигне. Потоа се зима слика од енд ефекторот и се зголемува фовто за да имаме поширока слика. Ориентацијата на енд ефекотрот се помстува за 180 по џ оска бидејки цамерати зима слика од з оската нагоре. Пубиллет ни врака три типа од сликата, основната ргб слика, длабоцината на пикселите и слика со сегментација каде објектие се сегментирани посебно. За да проблемот биде олеснет се користи сегментацијата каде знаеме дека со вредност 1 е легото на сликата. Со помош на нумѕу се зима позицијата на пикселите касе вредноста е 1 и врзиме средниа по двете оски за да го најдеме центарот на објектот во сликата. Следно ја роунднуваме посизијата до најблискиот пиксел и со помос на вградената функција rgbd\_2\_world() ја конверираме позицијата од пиксели и длабочината во таа точка во кординати на симулацијата. Таргет поцизијата ја поместуваме по з за 0.75 за точно поставување на гриперот. Потоа енд ефекторот оди на објектот, се симиња по з над него и со actuate\_attachment го затвораме гриперот преку код. На крај се подига раката на почетната позиција.

robot.move2point((0,0,2.5),(180,0,0))

camera = robot.capture\_image(fov=80,capture\_now=True)

h,w = np.mean((np.where(camera.seg == 1)), axis=1)

h,w = round(h),round(w)

target\_pos = camera.rgbd\_2\_world(w,h,camera.depth[h,w]) + [0,0,0.75]

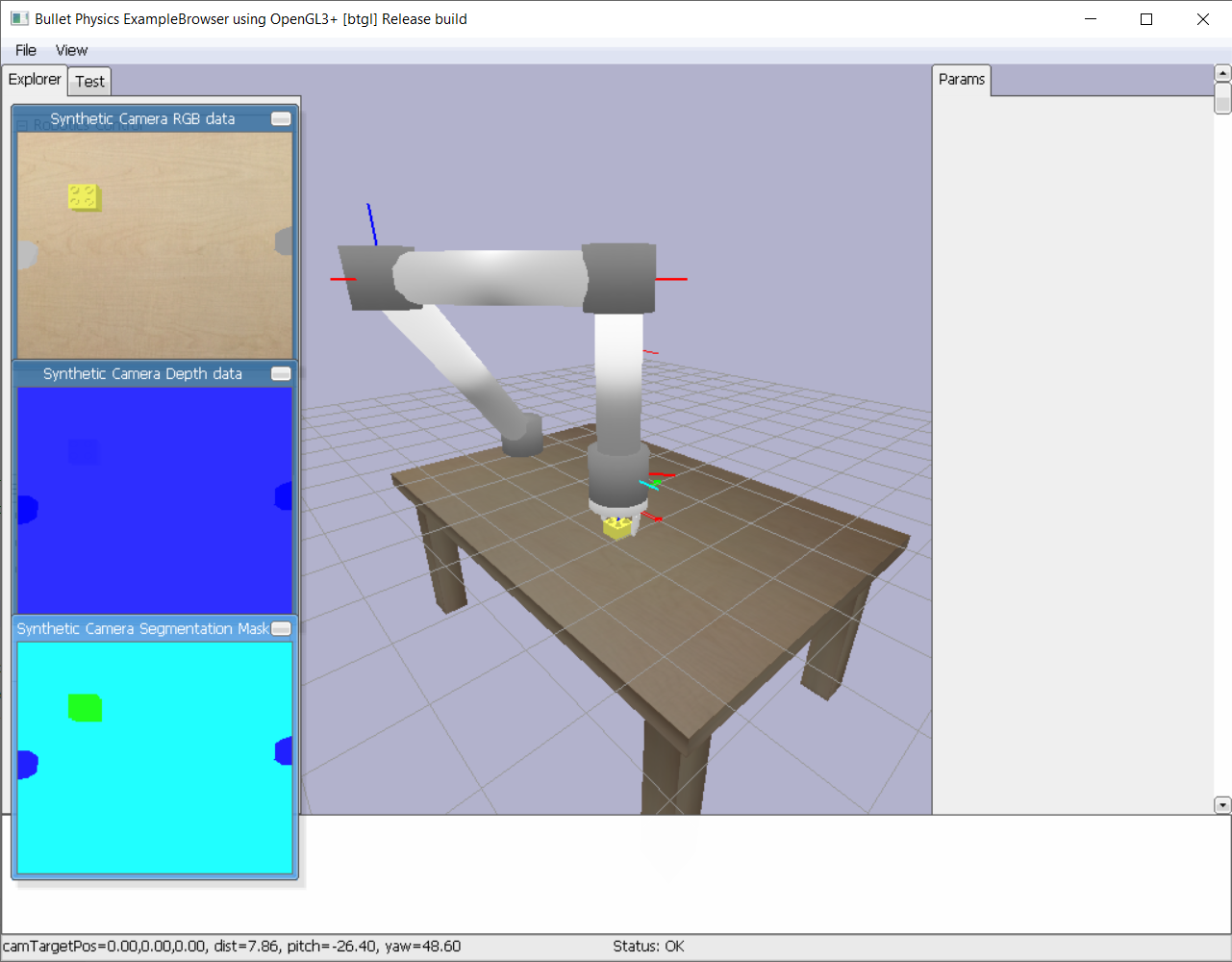
robot.move2point((target\_pos[0],target\_pos[1],2.5),(180,0,0))

robot.move2point(target\_pos,(180,0,0))

robot.actuate\_attachment(joint\_targets = robot.attachment\_close\_targets)

robot.move2point((0,0,2),(180,0,0))

Кога ке се земе слика од симулацијата од левата страна на гуи прозорецот се прекажуваат три прозорци со превју од земенеата слика.



## Популарни роботски раце

Со оваа јупитер тетратка ќе се разгледаат две примери од вистински роботски раце, како да се импортираат во пѕбуллет и контролираат. Се импортираат истите библиотеки и класи како прредходната тетратка.

### Franka Emirka panda роботска рака

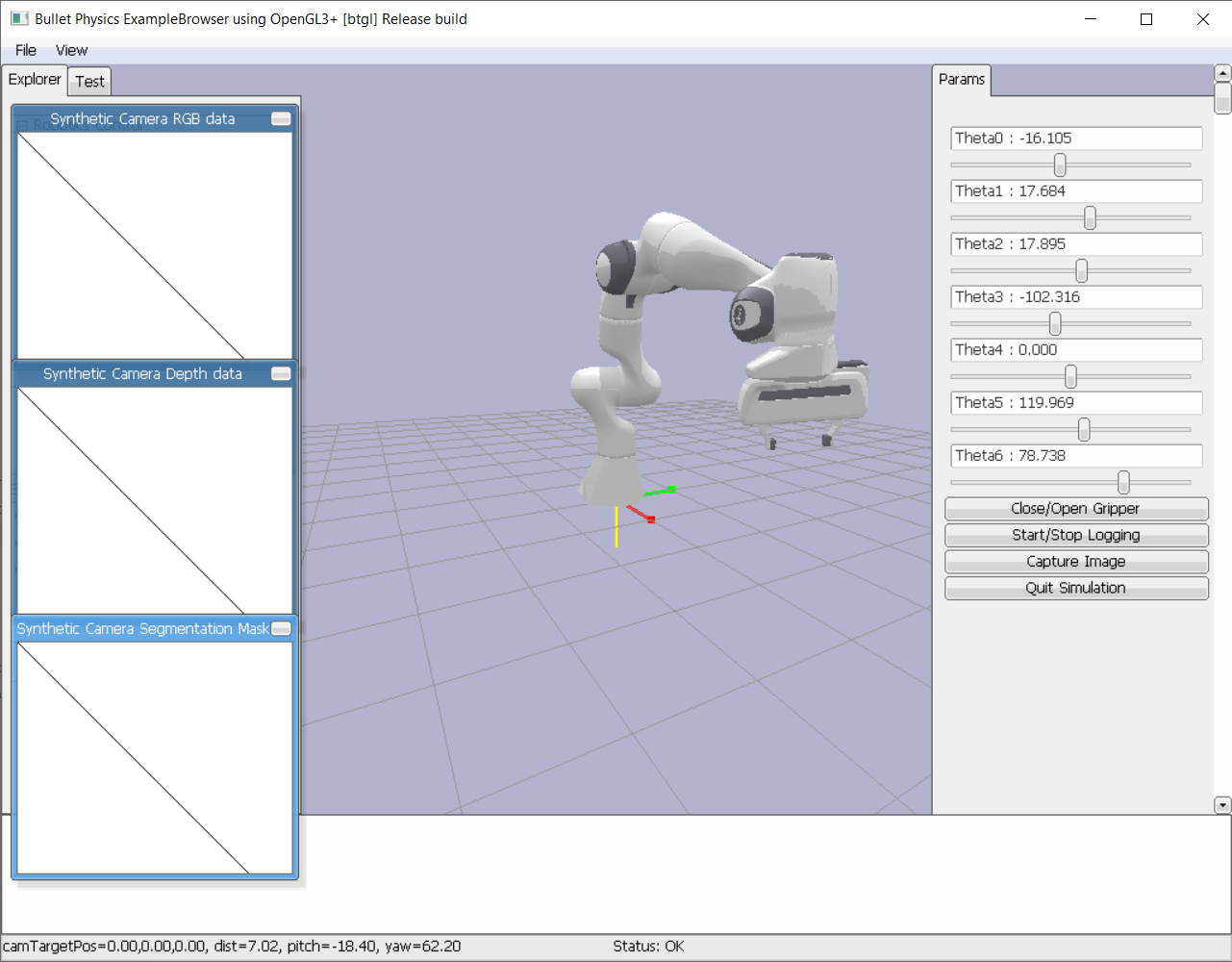
Прво е фрканка панда каде странските раце се импортираат преку import\_foreign\_robot() каде освен локацијата дополнително имаме и скалиранје за да се наместат димензии на раката. Исто така се поставуваат и таргетите за гриперот да се отвори и затвори.

robot = RobotArm()

robot.import\_foreign\_robot('robot\_arms/panda/panda.urdf')

robot.set\_attachment\_targets((0.2,0.2), (0,0))

robot.interact()



### xArm роботкса рака

Другата рака е од џарм и имамае повеке конфигурации, каде може да се видат во робот армс / џармс фолдерот. Прво креираме свет каде се полнат рандом објекти дадени од пѕбуллед и се наогаат во пѕбулеед дата . Потоа користиме инверзна кинематика и ја контролираме. Дополнително може да се постави и опсегот на слајдерите за позиција, во овој пример го поставуваме з да не оди во минус.

sim = PybulletSimulation()

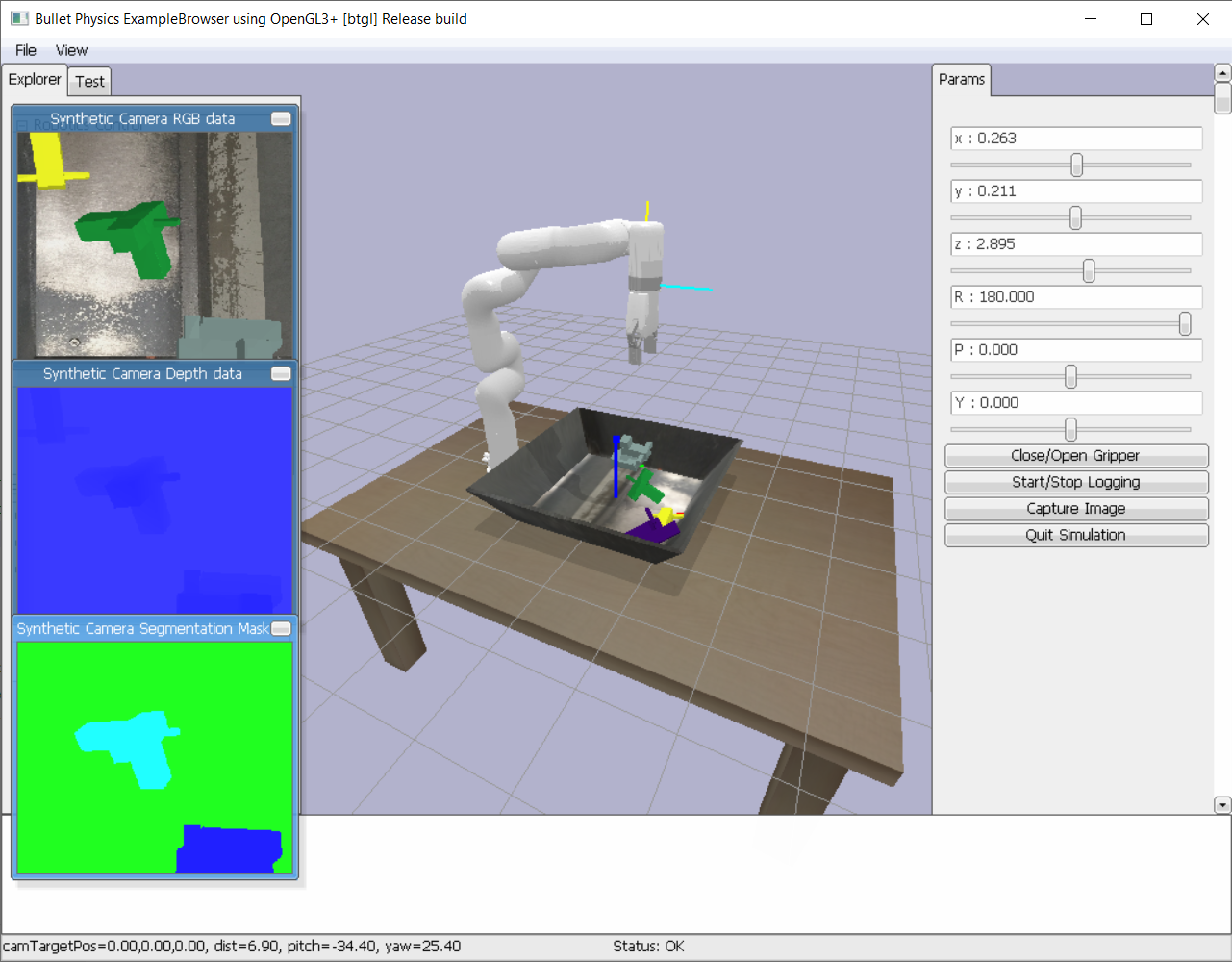
sim.load\_playground\_2()

robot = RobotArm((-2,0,0))

robot.import\_foreign\_robot('robot\_arms/xarm/xarm6\_with\_gripper.urdf')

robot.set\_attachment\_targets([0]\*6, [0.6981]\*6)

robot.interact('inverse',z\_range=(0,5,2))



## Mitsubishi роботска рака

Со оваа јупитер тетратка ја симулираме раката RV-2F-Q каде плус креираке програма за контрола на вистинската рака. Креарани се две верзии на раката со и без гриперот, засега програмата разботи без грипер бидејки не е усте целосно имплементиран на раката.

### Контрола на Mitsubishi роботската рака со гриппер

Креираме свет во кој се полни маса траѕ и сите цоммон објекти дадени од пубуллет. Дополнително се поставува и оффсетот на камерата за да бите добро поставена од почетокот на грипперот.

sim = PybulletSimulation()

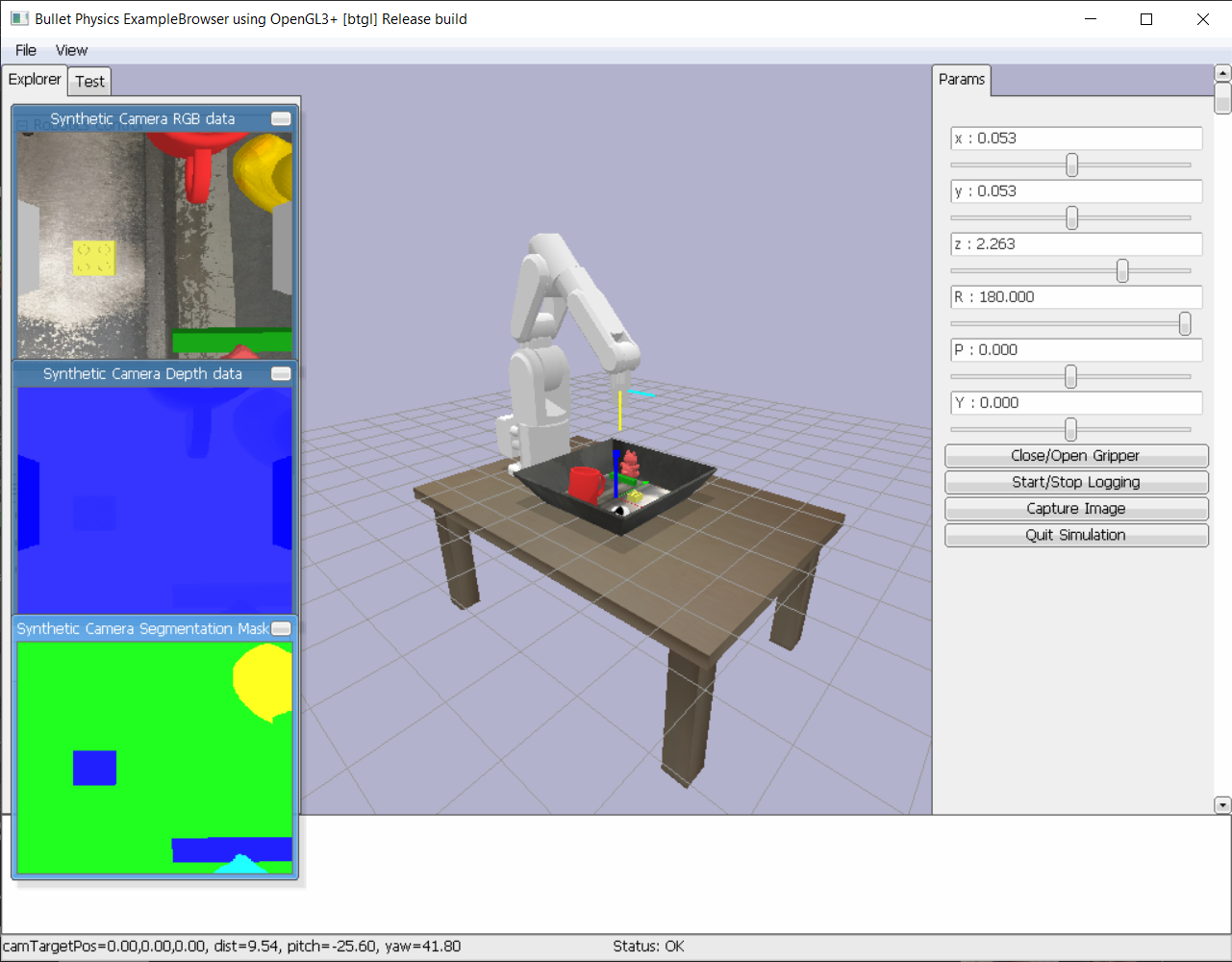
sim.load\_playground\_1()

robot = RobotArm((-2.2,0,0))

robot.camera\_offset = (0,0,0.02)

robot.import\_foreign\_robot('robot\_arms/mitsubishi/RV2FQG.urdf',8)

robot.interact('inverse')



### Поднесување на параметри на раката за контрола преку комадни

Креираме свет кој точно го реплицира во реалноста, не користиме динамика бидејки ке го користиме креирање на програма и секој чекор ке биде прецизен нема да чекаме раката да стигне до саканата позиција. Многу важно е да се постават и лимитите за до каде може раката да оди, во вој слуцај да не удри во масата и самата себе. На крај се ресетираат збголобовите за да помогне при нумеричката инверзна кинематика бидејки ако почне од нулта позиција сака да дојде кон другото решение но неможе бидејки имаме лимити. Може и опционално да се сетира и колку линијата за траекторијата ке пиде прикажана.

sim = PybulletSimulation()

sim.load\_table()

robot = RobotArm(use\_dynamics=False) robot.import\_foreign\_robot('robot\_arms/mitsubishi/RV2FQ.urdf',8)

robot.set\_position\_limits(1.8,3.5,-3,3,0.3,4)

robot.load\_robot()

robot.reset\_joints([0,0,2,0,0,0])

# robot.trajectory\_lifetime = 30 # set the line life to 30 seconds

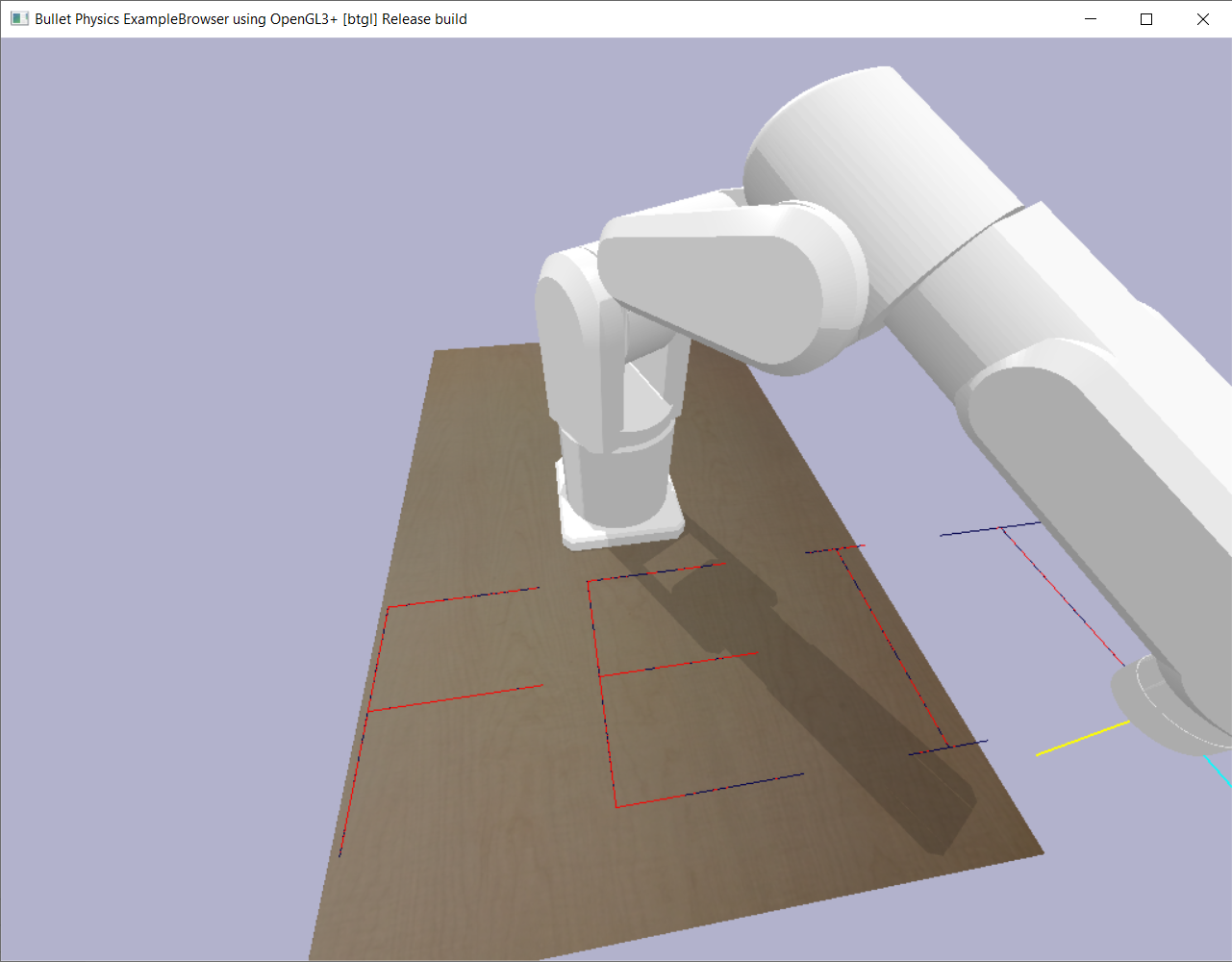
Следниот блок линии преставуваат сите опционалности како да се генерираат таргет точките. Најпросто движење е линеарното каде поставуваме почетна и крајна точка и посакуваната ориентациј која исто така може да се интерполира. Исто така воведено е и крижно интерполирање помеѓу двете точки каде дополнително се поставуваат и четирите параметри. Првите две се а и б за како да изгледа елипсата, а другите две се бинарни вредности со кои првиот одбира кое решение да се одбере, и од која старана да започне да црта. Исто така може да се да се сетираа и колку точки да се генерираат и рамнината на кои да се поректираат точките. На крај функцијата има дополнително и функција да започе со логирање на зглообовите. Како апстракција имам воведено функција да црта текст каде се користат движењата, засега има мал број на букви што се направени.

robot.move((3,-2,3,180,0,0),(2,3,2,180,0,0))

robot.move((3.8,0,0,180,0,0),(4,0,0,180,0,0),

(0,-45,0),'circular',60,param=(1,1,1,0),log\_move=True)

robot.write\_text('FEIT',[2.3,-1.5,2],(180,0,0),log\_text=False)



За крај може да се проверат зачуваните логови и со функцијата convert\_logs\_to\_prg() да се генерира програма која потоа може да се изврши на раката.

robot.replay\_logs('move\_circular')

robot.convert\_logs\_to\_prg('FEIT','feit')

## Кратки копчиња за pybullet

|  |  |  |
| --- | --- | --- |
| Акција | Операција | Инструкција |
| Отворете или затворете ги табовите за пребарување, тестирање и параметри | g | Притиснете g за да ги префрлите сите менија |
| Паузирај ја симулацијата за физичка | i | Притиснете i за да ја паузирате симулацијата |
| Дампинг на профилот | p | Притиснете p за ги зачувате логовите за користење на нишките |
| Вклучете ги светлата и сенките | s | Притиснете s за да ги вклучите/исклучите светлата и сенките |
| Вклучете ја визуелната геометрија | v | Притиснете v за да вклучите/исклучите визуелизацијата на геометриските лица |
| Вклучете ја жичната рамка | w | Притиснете w за да го вклучите/исклучите режимот на рамка |
| Излезете од апликацијата | Ecs | Притиснете Esc за да излезете од симулаторот |
| Премести објект | click + drag | Кликнете на објект во рамките на симулацијата за да примените сила |
| Ротирај го приказот | Ctrl + drag | Кога го држите копчето Alt или Control, левото кликнување и влечење на глувчето ја ротира камерата |
| Преглед на превод | Ctrl + middle click | Кога го држите копчето Alt или Control, среден клик и влечење на глувчето ја поместува камерата |
| Зумирајте/одзумирате приказ | mouseweel | Користете го глувчето за да зумирате и одзумирате |
| Зумирајте/одзумирате приказ | Ctrl + right click | Кога го држите копчето Alt или Control, кликнете со десното копче на глувчето и повлечете ја камерата за зумирање и одзумирање |
| Вклучете го серверот за физика за зачувување слики | F1 | Притиснете F1 за да го префрлите серверот за физика за фрлање слики (континуирано ги зачувува сликите од екранот на симулаторот) |
| Зачувај „testFile.bullet“ | F3 | Притиснете го копчето F3 за да зачувате „testFile.bullet“ фајл од светот |

# Надградување на софтверскиот пакет

Проектот е направен од старт да биде модуларен каде може многу работи да се напреднат. Лесно може да се додадат нови алакти и вистински роботски раце, исто така да се додадат и нови објекни во симулацијата. Има и многу повеке функционалности што може да се наградат како и примери користејки ја симулацајата. Во следните подточки ќе биде објаснето во детали како да се имплементираат новите функционалности.

## Додавање на нова алатка

За да се креира нов алат што ќе се додаде на роботкстака рака, треба да се следи следната конвенција. Се креира нова папка со името на алатот во папката /attachments и се ставаат сите датотеки во неa. Главниот urdf треба да биде со исто име со папката и првата врска во датотеката треба да се нарекува base за да може да се рефренцира со крајниот зглоб на раката. Следно треба да се провери дали алатот е компатибилен со функцијата actuate\_attachment(), засега функцијата е кодирана за отворање и затворање на грипер. За слични грипери, може само да се искористи функијата set\_attachment\_targets() со која се поставуваат саканите позииции на гриперот кога е отворен и затворен.

## Додавање на новa роботска рака

Кога се додава нова роботка рака треба да се провери дали патеката на визуелните датотеки е правилно венсени во главниот фајл. Важно е да се остранат и сите xacro макроа бидејки работат со pybullet, а има многу раце така конфигурарано онлајн за олеснување при работа со нив. Има два примери panda и xarm во директориумот robot\_arms/ како примери за како точно треба да се имплементира новата рака. Ако раката на роботот има додаток, треба да се преименува зглобот што го поврзува со раката во attachment\_joint за функцијата find\_joint\_ids() правилно да ги идентификува сите згобови на роботската рака.

## Креирање на нов свет во pybullet

Користејќи ја класата PybulletSimulation, може да се направи нова функции за додавање објекти во симулацијата. Засега само се користат urdf датотеките обезбедени од pybullet каде се направени основни објекти како работна маса, треј и сите вообичаени објекти пронајдени во директоријата pybullet\_data.

## Нови функционалности на софтверскиот пакет

Проектот има многу добра основа и функционалности за креирање на комплициран свет, но секогаш ќе има подобрувања и нови функционалности во pybullet што ќе треба да се имплементираат и тука. Затоа целиот проек е и отворен за сите за да може да се продолжи со неговиот девелопмент и сите придонесѕвања се добредоени. Имаме поставено листа од можни имплементации што уште не се имплементирани:

* Опција за внесување на инерцијални вредности на линковите на раката
* Имплементација на сферични зглобови
* Други типови на интерполација
* Да се додадат и останатите букви во write\_letter()
* Имплементација на инверзна динамика
* Нова класа за креирање и контрола на мобилни роботи
* Нова класа за креирање и конторла на квадропед робот

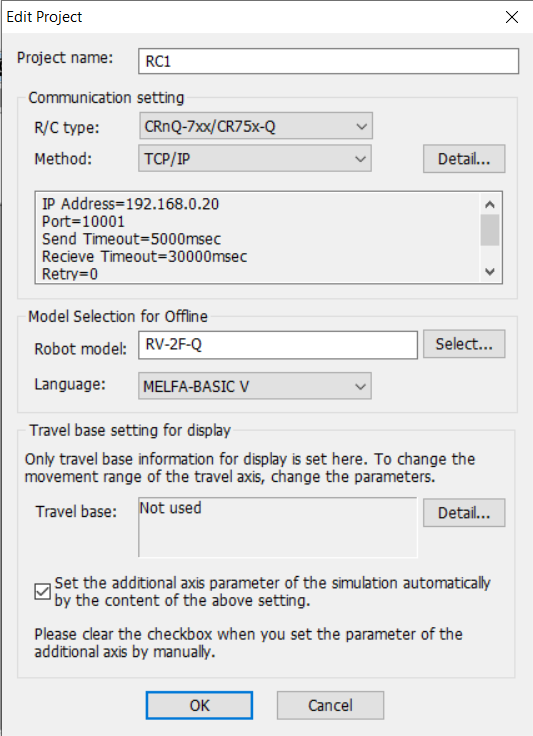
## Нови примери користејки го софтверскиот пакет

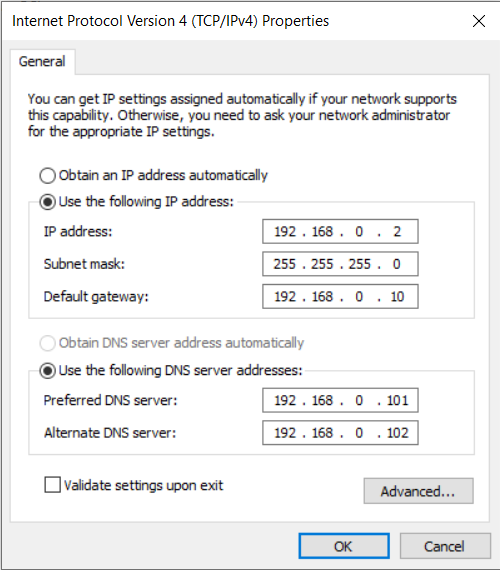
Со трите јупитер тетратки воглавно се поминати сите основни функционалности на софтверскиот пакет, но има многу моножноски за креирање на многу покомплицирани сценарија. Подоле е поставено листа на можни имплементации на нови примери:

* Покомплициран систем за детекција и зимање на објекти
* Имплементација на мобилана роботкса рака
* Имплементирање на tossingbot
* Користење на машинско учење за контрола на раката

# Конектирање на CR750-Q со RT Toolbox 2

Комуникацијата помеѓу компјутерот и контролерот CR750-Q се врши преку етернет кабел. Кабелот е поврзан со plc интернет-портата на CR750-Q контролерот и во интернет портата на компјутерот. Параметрите на контролорот на роботот за TCP/IP комуникација се веќе поставени. Прво во Windows треба да се отиде во Control Panel > Network and Internet > Network and Sharing Center > Change adapter settings > Select the Ethernet Properties > Choose the TCP/IPv4 properties и да се постават својствата на IPv4 како на сликата 1неш. Во програмата RT Toolbox 2 се креира нов проект и се копираат параметрите од сликата подолу. Потоа од менито на RT Toolbox 2 се отвора Online, и како под опција се селектира пак Онлине со која програмата почиња да воспоставува комуникација со контролерот. Се појавува нов прозорец кој не информира за статусот на врската и ако се е подесено како треба, статусот на линијата треба да покаже конектирана и да свети сино.





# Архитектура на системот

## Хиерархија помеѓу класите

## Хиерархија на класата PybulletSimulation

## Хиерархија на класата Camera

## Хиерархија на класата DH2Urdf

## Хиерархија на функцијата interact

## Хиерархија на контрола преку команди

## Хиерархија за креирање на модуларна роботска рака

## Модули за симулација на роботската рака

# Докс

# Заклучок

# Референци

1. <https://docs.conda.io/en/latest/miniconda.html>