คู่มือวิชา Robot Operating System (ROS)

จัดทำโดย

นาย ชินวัตร ซาเล

นาย ปณชัย ทองไพรวรรณ

นาย ชนาเทพ ศิลาอาสน์

จาก

คณะ วิศวกรรมศาสตร์ ภาควิชา วิศวกรรมการผลิตและหุ่นยนต์

สาขาวิชา วิศวกรรมหุ่นยนต์และระบบอัตโนมัติ

มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ(KMUTNB)

นำเสนอ

สถาบันพัฒนาบุคลากรสาขาเทคโนโลยีการผลิตอัตโนมัติและหุ่นยนต์

คู่มือนี้จัดทำขึ้นเพื่อช่วยให้ผู้อ่านได้เข้าใจ ROS มากขึ้น

สิ่งที่ผู้สอนและผู้เรียนต้องเตรียม

- คอมพิวเตอร์/โน๊ตบุค
- อินเตอร์เน็ต
- Ubuntu

แผนการสอน

บทที่1 ลงโปรแกรม ROS และอธิบายพื้นฐาน ROS

บทที่2 ลงโปรแกรม Python และอธิบายพื้นฐาน Python

บทที่3 เรียนการทำ database clound และการสร้าง GUI

บทที่4 ใช้ arduino สื่อสารกับระบบ ROS และก็มีการสร้าง simulation ใน tinker cad

บทที่5 ได้ใช้ทั้งหมดที่เรียนกับ ROBOT ARM

บทที่1

ROS

เราจะมาทำความเข้าใจก่อนว่าRobot Operating System หรือว่า ROS นั้นคืออะไรเอาไปใช้ได้ยังไงแล้วประโยชน์คืออะไรสามารถนำไปใช้ในชีวิตประจำได้หรือไม่

Robot Operating System(ROS) เป็นระบบที่สร้างขึ้นเพื่อทำให้เกิดความยืดหยุ่นในการเขียนซอฟต์แวร์ควบคุมหุ่นยนต์ ซึ่งใน ROS จะรวบรวมชุดเครื่องมือและชุดคำสั่งต่างๆที่จำเป็นในการพัฒนาหุ่นยนต์เอาไว้ ซึ่งสิ่งต่างๆเหล่านี้จะลดความยุ่งยากในการสร้างในการพัฒนาหุ่นยนต์ที่มีความซับซ้อน และทำให้มีประสิทธิภาพในการพัฒนาหุ่นยนต์หลากหลายรูปแบบ

ROS ได้รับการพัฒนาจากหลากหลายหน่วยงาน สำหรับการพัฒนาหุ่นยนต์หลากหลายรูปแบบ แม้ว่าข้อมูลทั้งหมดจะอยู่บนเซิร์ฟเวอร์เดียวกันเพื่อให้ผู้คนที่สนใจเข้าถึงได้ง่ายในช่วงหลายปีที่ผ่านมา แต่ไม่นานรูปแบบของการเก็บข้อมูลบนเซิร์ฟเวอร์แยก ได้ถูกนำมาใช้เป็นจุดแข็งสำหรับกลุ่มคนที่ใช้ ROS โดยพวกเขาสามารถที่จะเป็นเก็บ source code ของตัวเองบนเซิร์ฟเวอร์ของตัวเองได้ และยังรักษาความเป็นเจ้าของและการควบคุมการเข้าถึง source code ได้เต็มรูปแบบ โดยไม่ต้องขออนุญาตจากใคร ถ้าหากพวกเขาต้องการเผยแพร่ข้อมูลเหล่านั้นต่อสาธารณะพวกเขาสามารถทำได้และเป็นที่ยอมรับในหมู่คนที่ใช้ ROS ด้วยกัน พวกเขาอาจได้รับค่าตอบแทนที่พวกเขาสมควรได้รับสำหรับความสำเร็จของพวกเขา และได้ประโยชน์จากข้อเสนอแนะทาง

เทคนิคเพื่อการปรับปรุงซอฟต์แวร์ของพวกเขาให้ดีขึ้น เช่นเดียวกับโครงการซอฟต์แวร์โอเพนซอร์สต่างๆ

ในขณะนี้ชุมชนของผู้ใช้ ROS ประกอบด้วยผู้ใช้นับหมื่นทั่วโลก ตั้งแต่โครงการงานอดิเรกเล็กๆ ไปจนถึงระบบอัตโนมัติอุตสาหกรรมขนาดใหญ่ ล้วนแล้วแต่ใช้ ROS ในการสร้างระบบของหุ่นยนต์

ROS Concept

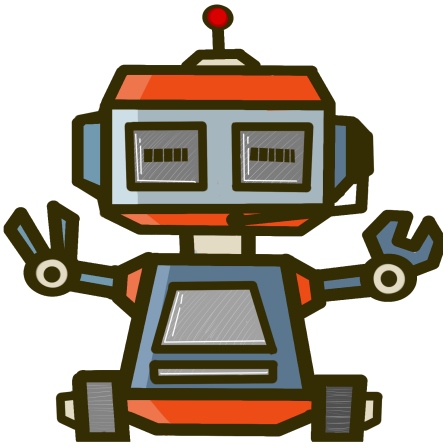
ระบบ ROS จะแบ่ง Concept ของระบบออกเป็น 3 ส่วน คือ

1. ROS Computation Graph Level

คอนเซ็ปของระบบ ROS ก็คือการแบ่งโมดูลการทำงานของหุ่นยนต์ออกเป็นส่วนๆ แล้วให้แต่ละส่วนส่งข้อความคุยกันผ่านทาง Topic ต่างๆในรูปแบบของ Peer-to-Peer กับส่วนอื่นๆ โดยจะมี ROS Master เป็นตัวบริหารจัดการทำงานของระบบทั้งหมด ดังนั้นจะมีคำศัพท์ที่ควรจะรู้ไว้ก็คือ nodes, Master, Parameter Server, messages, services, topics, และ bags

2. ROS Filesystem Level

ระบบจัดเก็บไฟล์เป็นคอนเซ็ปหลักอีกอย่างหนึ่งของระบบ ROS ซึ่งจะเป็นแหล่งทรัพยากรในการเรียกใช้งานของแต่ละโหนดแต่ละโมดูล โดยจะมีส่วนที่สำคัญดังต่อไปนี้



3. ROS Community Level

ในส่วนคอนเซ็ปของระบบชุมชนผู้ใช้ ROS จะเป็นแหล่งที่ใช้ในการศึกษา แลกเปลี่ยนความรู้ ดาวน์โหลด packages ต่างๆมาใช้งาน อีกทั้งยังใช้เป็นแหล่งเผยแพร่ packages ที่เราออกแบบ

ซึ่งคู่มือนี้เราจะเป็นการเน้นเนื้อหาBasicต่างๆของ ROS ให้ผู้อ่านมีความรู้ความเข้าใจในพื้นฐานของ ROS ดังนั้นเราะเน้นไปทางด้านของ ROS Computation Graph Level โดยจากบทความconceptด้านบน เราจะมีการอธิบายเพิ่มเติมต่อคำศัพท์ต่างของROSให้มีความเข้าใจได้มากขึ้น

- **Node :**เป็นกระบวนการที่ใช้ในการดำเนินการโปรแกรมต่าง ซึ่ง ROS ได้รับการออกแบบให้เป็นระบบแบบแยกส่วนในระดับที่ละเอียด ระบบควบคุมหุ่นยนต์มักจะประกอบด้วยหลายโหนด ตัวอย่างเช่นหนึ่งโหนดควบคุม laser range-finder, หนึ่งโหนดควบคุมมอเตอร์ล้อ, หนึ่งโหนดทำการแปลงค่าต่างๆ, หนึ่งโหนดทำการวางแผนเส้นทาง, หนึ่งโหนดให้มุมมองกราฟิกของระบบและอื่น ๆ โหนด ROS ถูกเขียนด้วยการใช้ไลบรารีของ ROS เช่น roscpp หรือ rospy
- **Master :** ROS Master จัดเตรียมการลงทะเบียนชื่อและค้นหาองค์ประกอบต่างๆใน ROS หากไม่มี Master โหนดจะไม่สามารถค้นหาซึ่งกันและกันเพื่อแลกเปลี่ยนข้อความหรือเรียกใช้บริการได้
- **Parameter Server :** พารามิเตอร์เซิร์ฟเวอร์จะอนุญาตให้จัดเก็บข้อมูลในตำแหน่งส่วนกลาง ซึ่งเป็นส่วนหนึ่งของ Master
- **Message :** โหนดสื่อสารกันโดยส่งข้อความ โดยอาจจะคุยในรูปแบบของข้อความ ค่ามาตรฐาน (จำนวนเต็ม, จุดลอย, บูลีน, ฯลฯ) ข้อมูลอาร์เรย์ทั่วๆไป รวมไปถึงข้อมูลอาร์เรย์ที่มีความซับซ้อน

- **Topics :** ข้อความจะถูกส่งผ่านระบบการขนส่งที่มีการ publish / subscribe โหนดที่ส่งข้อความจะเรียกว่า publishing ชื่อที่ระบุใน Topic จะใช้เพื่อระบุเนื้อหาของข้อความ โหนดที่สนใจในข้อมูลบางประเภทจะทำการ subscribe หัวข้อที่ต้องการ อาจมีการ publish / subscribe หลายรายพร้อมกันสำหรับหัวข้อเดียวและโหนดเดียวอาจ publish / subscribe ได้หลาย Topics โดยทั่วไป publish / subscribe จะไม่ได้ตระหนักถึงการมีอยู่ของกันและกัน กล่าวอีกนัยคือการแยกกันผลิตข้อมูลตามหน้าที่ของตนเอง
- **Services :** รูปแบบการ publish / subscribe เป็นแนวคิดในการสื่อสารที่ยืดหยุ่นมาก แต่การขนส่งทางเดียวอย่างต่อเนื่องนั้นไม่เหมาะสม บางครั้งจึงมีการใช้บริการ Request / reply กำหนดการส่งข้อความ โดยสร้างข้อความหนึ่งสำหรับการร้องขอและอีกหนึ่งสำหรับการตอบกลับ เพื่อส่งข้อความที่ร้องขอไปยังโหนดที่ร้องขอ
- **Bags :** เป็นรูปแบบสำหรับการบันทึกและเล่นข้อมูล ข้อความใน ROS Bags เป็นกลไกสำคัญในการจัดเก็บข้อมูลเช่นข้อมูลเซ็นเซอร์ที่อาจรวบรวมได้ยาก แต่จำเป็นสำหรับการพัฒนาและทดสอบอัลกอริทึม

สามารถอ่านเพิ่มเติมได้ที่ https://wiki.ros.org/

ต่อไปเมื่อเรารู้และเข้าใจเกี่ยวกับ ROS แล้วต่อไปเราจะมาทดลองใช้ ROS กันโดยขั้นแรก เราจะมาติดตั้งตัวของ ROS ก่อน เนื่องจากระบบนี้จำเป็นที่ต้องติดตั้ง ระบบ Linux กันสักก่อน มีให้เลือกสองรูปแบบUbuntu & Debian(สำหรับการติดตั้งมันจะขึ้นกับระบบ Version ROS ด้วยนะ) สำหรับการติดตั้งระบบนี้จะทำบนส่วนที่เป็น Terminal ของ Linux เขียนตามได้เลย

```
isudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
$(lsb_release -sc) main" >
/etc/apt/sources.list.d/ros-latest.list'
```

```
sudo apt install curl
```

```
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc |
sudo apt-key add -
```

หลังจาก โหลดสคริปไฟล์อัพเดตเสร็จ ก็ทำการ Update Linux ดังนี้

```
sudo apt update
```

ขั้นตอนถัดไป มีการติดตั้งโปรแกรมหรือฟังก์ชั่นเสริมต่างๆ ให้เลือกสำหรับผู้เริ่มต้นผมขอแนะนำลงไปทั้งหมดเลยครับ

```
sudo apt install ros-noetic-desktop-full
```

```
sudo apt install ros-noetic-rosserial-arduino
```

ขั้นตอนถัดไป มีการติดตั้งโปรแกรมของPython เช็ค Python และติดตั้งเพื่อนำมาใช้กับระบบของ ROS สำหรับการใช้งานสามารถเขียนได้ทั้งสองภาษา คือ **Python & C++** ถ้าต้องการรันบน Python ก็จะต้องติดตั้งอันนี้ด้วยครับ

```
python3 --version
```

```
sudo apt install python3-pip
```

```
source /opt/ros/noetic/setup.bash
```

```
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

```
sudo apt install python3-rosdep python3-rosinstall
python3-rosinstall-generator python3-wstool build-essential
```

ขั้นตอนสุดท้าย เป็นขั้นตอนตรวจสอบระบบ ROS Source ต่างๆ

```
sudo rosdep init
```

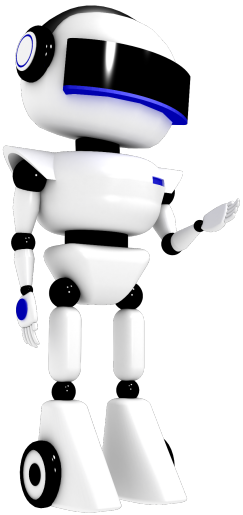
```
rosdep update
```

ขั้นตอนการสร้าง Enviroment และ ROS Workspace

มาเริ่มสร้าง catkin workspace กันเลย ด้วยคำสั่งดังนี้ :

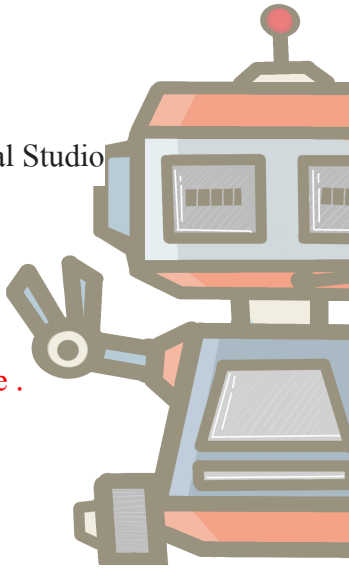
`mkdir catkin_ws`
`cd catkin_ws`
`mkdir src`
`catkin_make`
`source ~/catkin_ws/devel/setup.bash`
`gedit ~/.bashrc`
`cd src`
`catkin_create_pkg my_project rospy turtlesim rosserial_arduino`
`cd ..`
`catkin_make`
`sudo snap install code --classic`

if you want to know more information:
http://wiki.ros.org/noetic/Installation/Ubuntu



บทที่2

Python



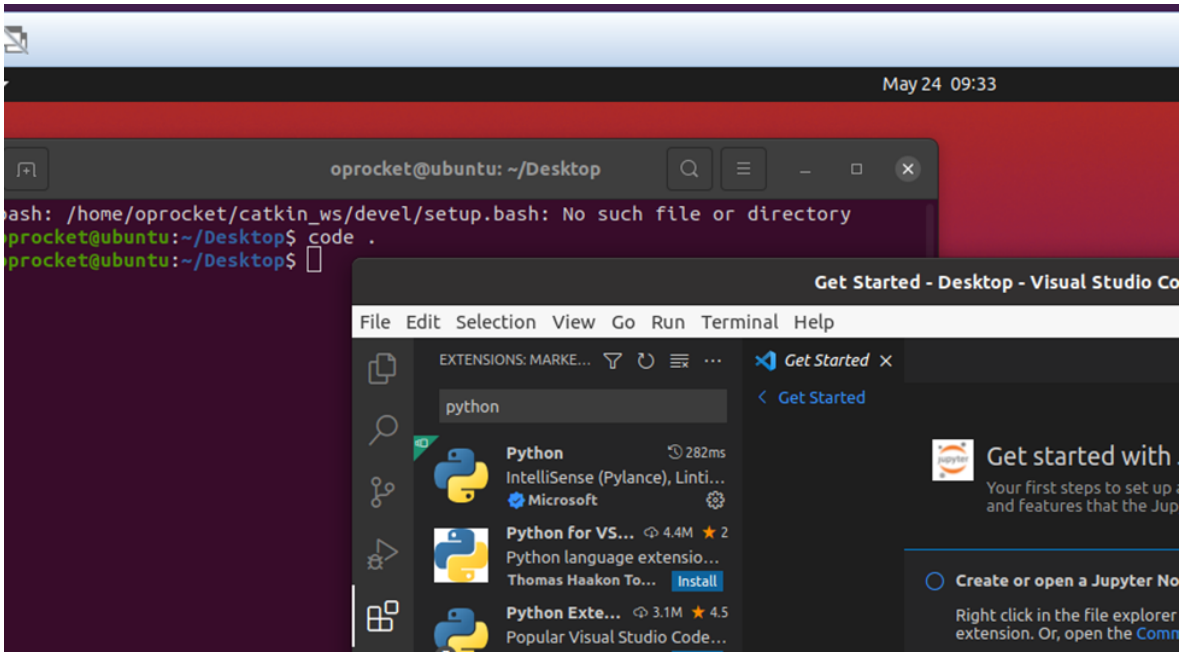
เริ่มแรกเราจะสร้าง pythonใน ubuntuโดยขั้นแรกเราจะทำการโหลดโปรแกรม Visual Studio Code ผ่านทาง terminal (Ctrl + Alt + T) โดยคำสั่งพิมใน terminal

: sudo snap install code – classic

หลังจากโหลด visual studio เสร็จแล้ว จะเปิดโปรแกรมจาก terminal ด้วยคำสั่ง code .

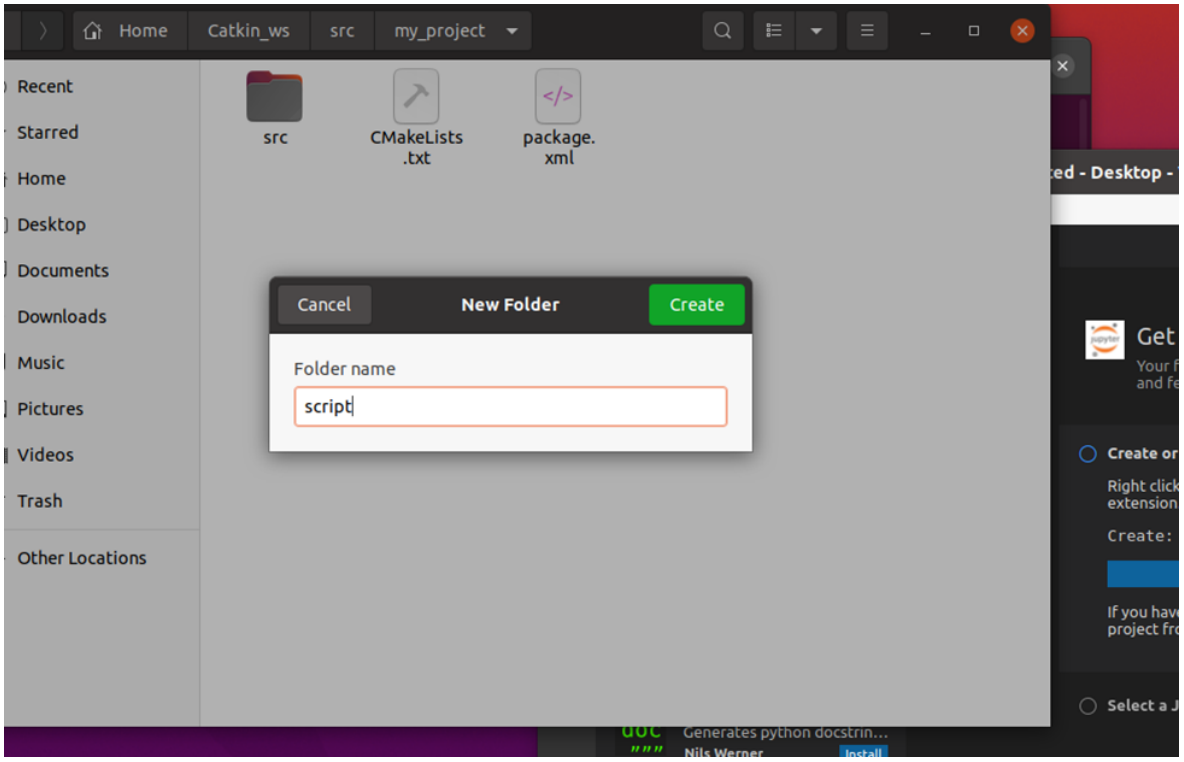
หลังจากเปิดโปรแกรมมาแล้วให้ติดตั้ง python ในโปรแกรม visual studio โดยไปที่ environment แล้ว search ว่า python

หลังจากนั้น install python มาดังภาพด่านล่าง



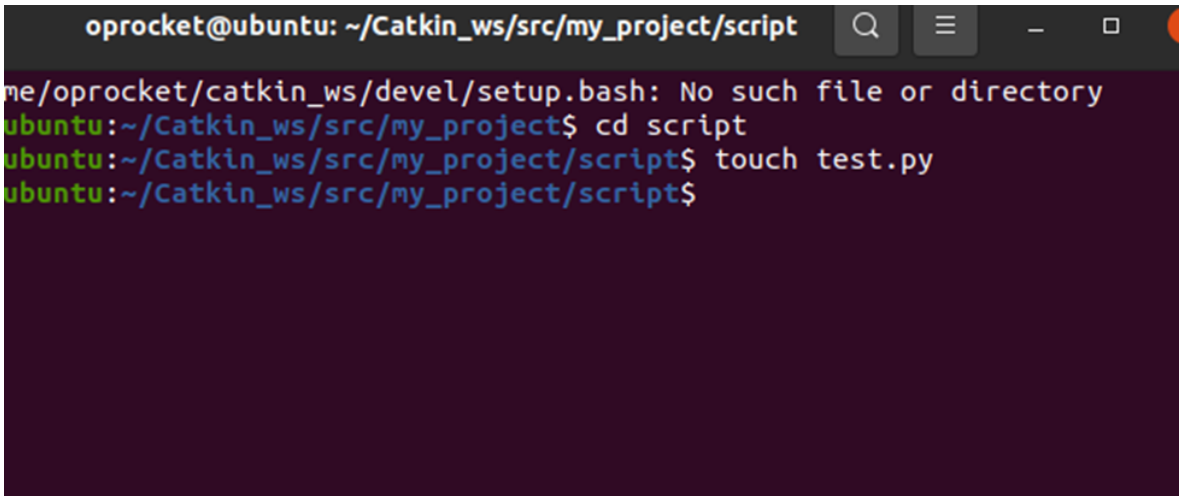
คำสั่งเปิดใช้งาน vscode = code .

หลังจากนั้นเราจะสร้าง folder script เพื่อที่จะใส่ไฟล์python เราสร้างขึ้นว่า script ตามรูปภาพด้านล่าง



ภาพแสดงการสร้าง โฟลเดอร์

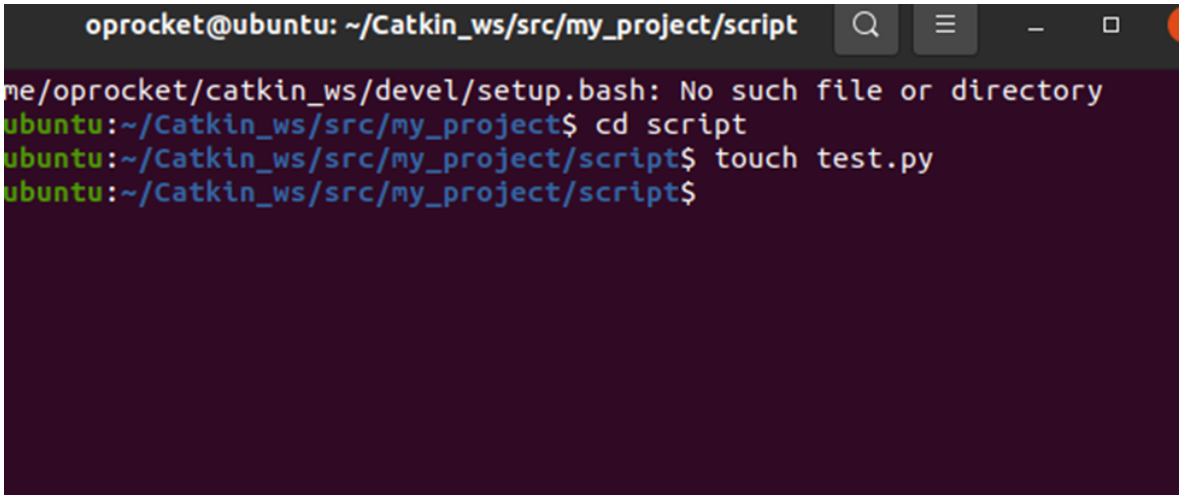
หลังจากนั้นเราจะสร้างไฟล์ .py โดยสร้างจาก terminal ด้วยคำสั่ง touch ตามด้วยชื่อไฟล์.py ตัวอย่างเช่น touch test.py ตามรูปด่านล่าง



คำสั่งสร้างไฟล์

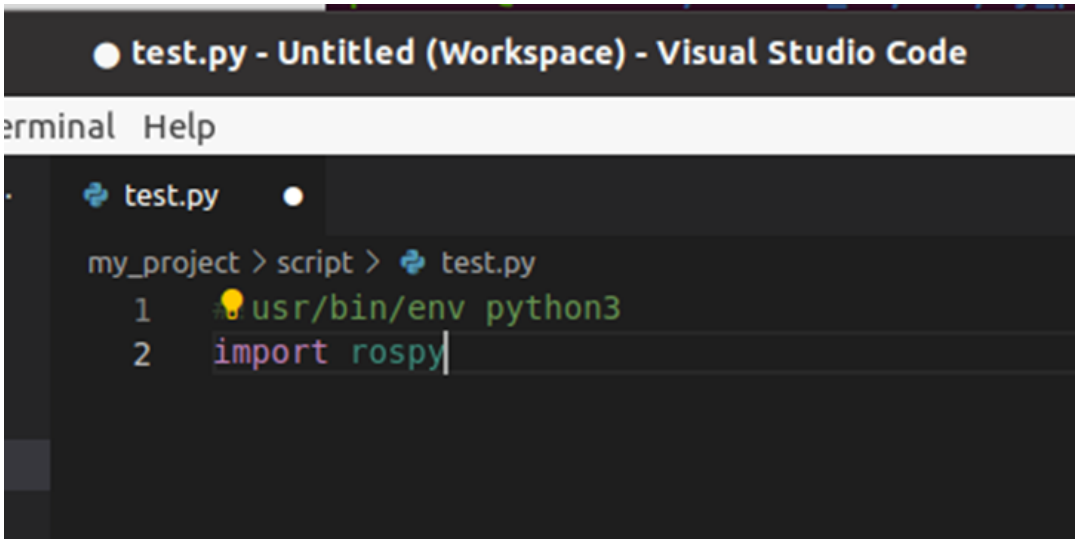
เมื่อมีไฟล์.pyเรียบร้อยแล้วหลังจากนั้นเราจะเขียนลงpythonลงไปไฟล์นั้นผ่านโปรแกรม visual studio โดยเปิดไฟล์เราสร้างขึ้นมาจากนั้นหัวข้อให้พิมว่า

#!/usr/bin/env python3



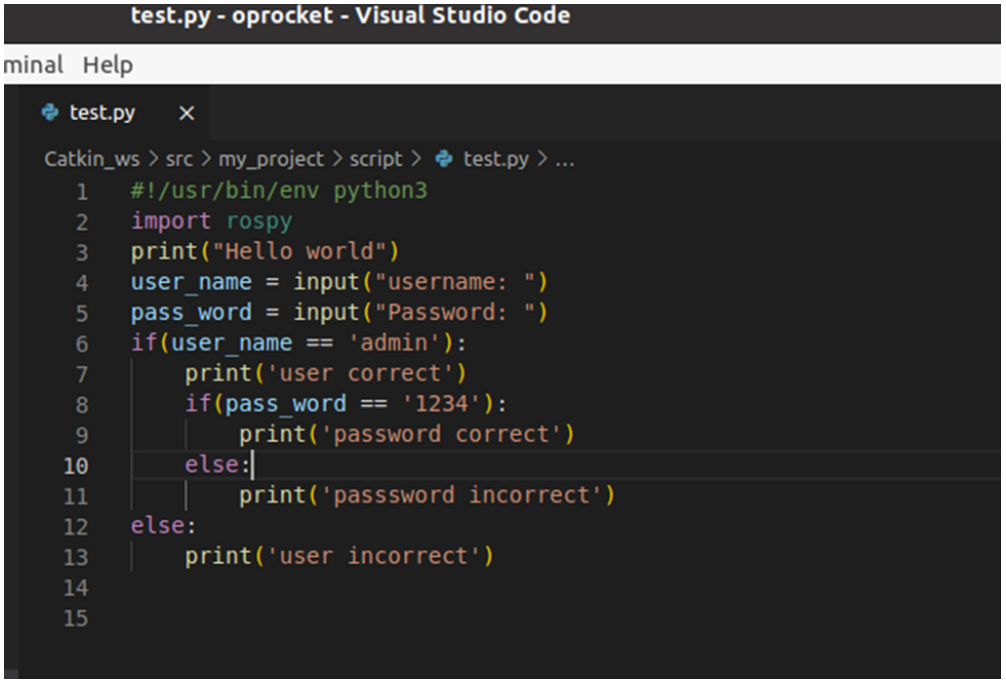
ภาพการสร้างไฟล์ python

เมื่อไหร่ก็ตามที่เราต้องการจะใช้เพื่อใช้ร่วมงานกับ ROS เราจะใส่หัวข้อ import rospy ขึ้นเสมอ ไม่งั้นจะไม่มีคำสั่งในการเขียนโปรแกรมผ่านทาง ROS



คำสั่งที่ต้องมีตลอดเมื่อใช้ ROS คู่กับ python

เราจะทำการทดลองเขียน python เป็นคำสั่ง basic ดูว่าเราจะรันได้ไหม

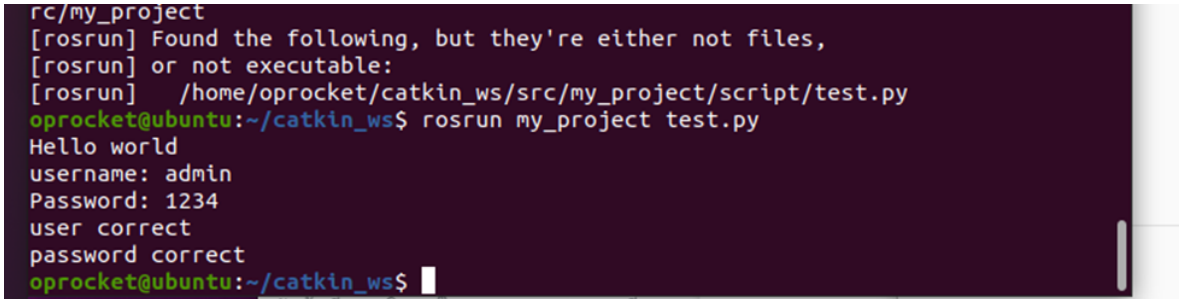


ภาพทดสอบว่าโปรแกรมทำงานได้ไหม

โดยการรันเราจะทำการรันไฟล์ python ที่เราเขียนผ่านทางตัวของ terminal ของubuntu

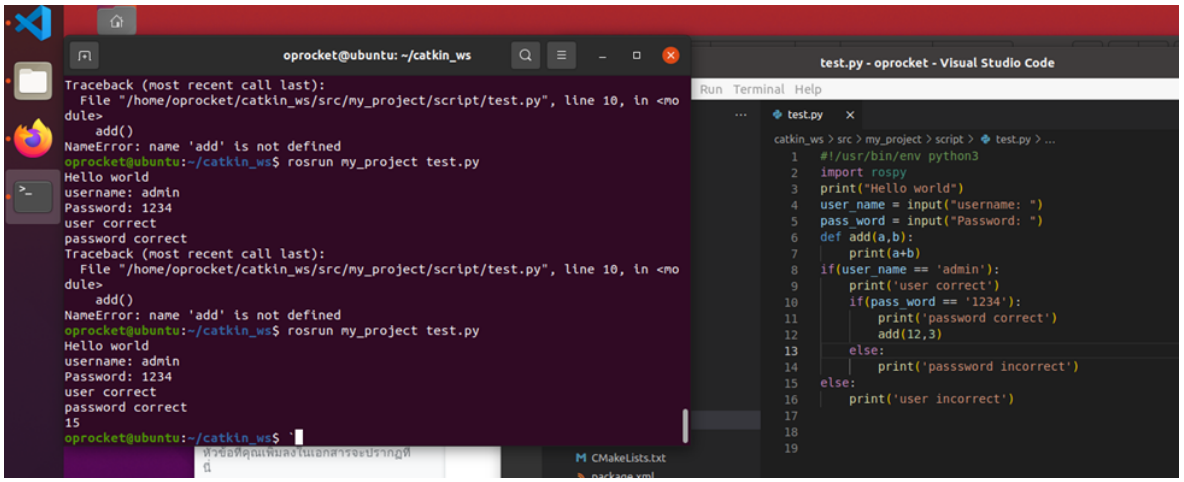
ด้วยการรันผ่าน package ที่เราสร้างขึ้นมา โดยคำสั่ง rosrun ตามด้วยชื่อแพคเกจของเราแล้วตามด้วยชื่อไฟล์ py

ตัวอย่าง rosrun my_project test.py



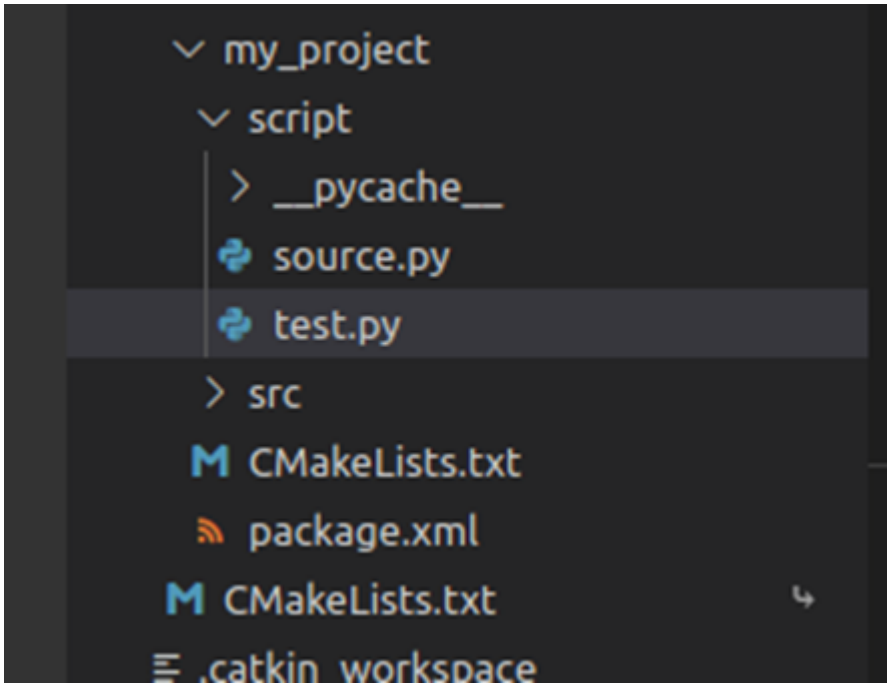
ตัวอย่าง rosrun my_project test.py

Code python ตัวอย่างสำหรับ run ในterminal แต่เขียนในcodeในvisual studio

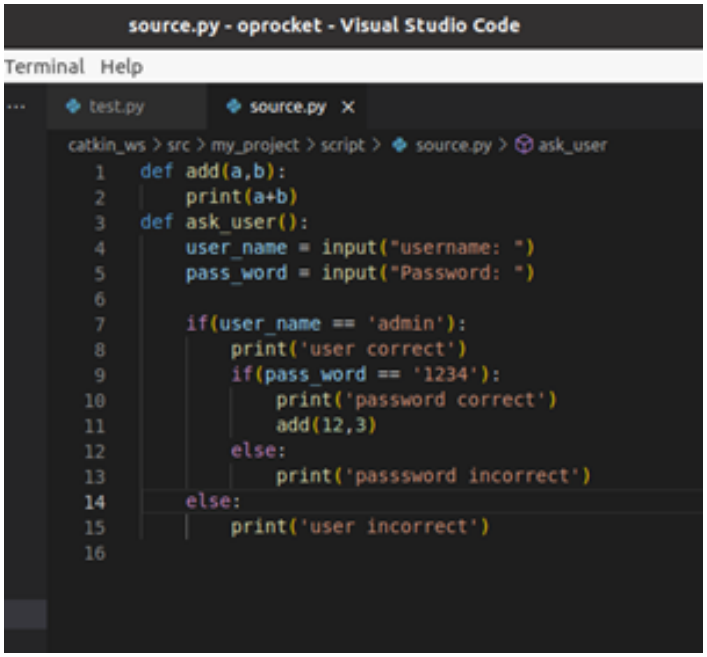


Code ตัวอย่าง
การสร้างไฟล์ที่ชื่อ source.py

ต่อไปเราจะลองสร้างการทำไฟล์แยกแต่สร้างความสัมพันธ์กันได้ โดยขั้นแรกเราจะสร้างไฟล์ .py ใหม่ ตัวอย่างเราจะสร้างไฟล์ชื่อ source.py

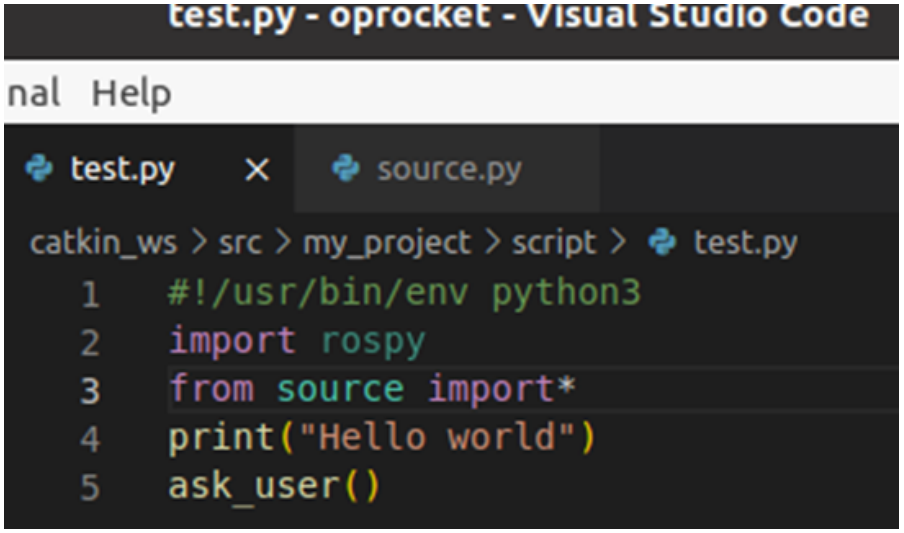


ขั้นต่อไปเราทำการสร้างตัวฟังก์ชันที่ชื่อว่า add และ ask_user เพื่อเรียกใช้โดยการเขียน def ใน source.py โดยในตัวอย่างเราจะมีฟังก์ชัน 2ตัวเพื่อเรียกค่าไปในตัวของ test.py



การสร้างฟังก์ชันที่ชื่อว่า add

ในตัว test.py จะเรียกค่าdefจาก source.py โดยพิมคำสั่ง from … แล้วพิมตัวแปลของ def ที่เราสร้าง เป็นการเสร็จสิ้น



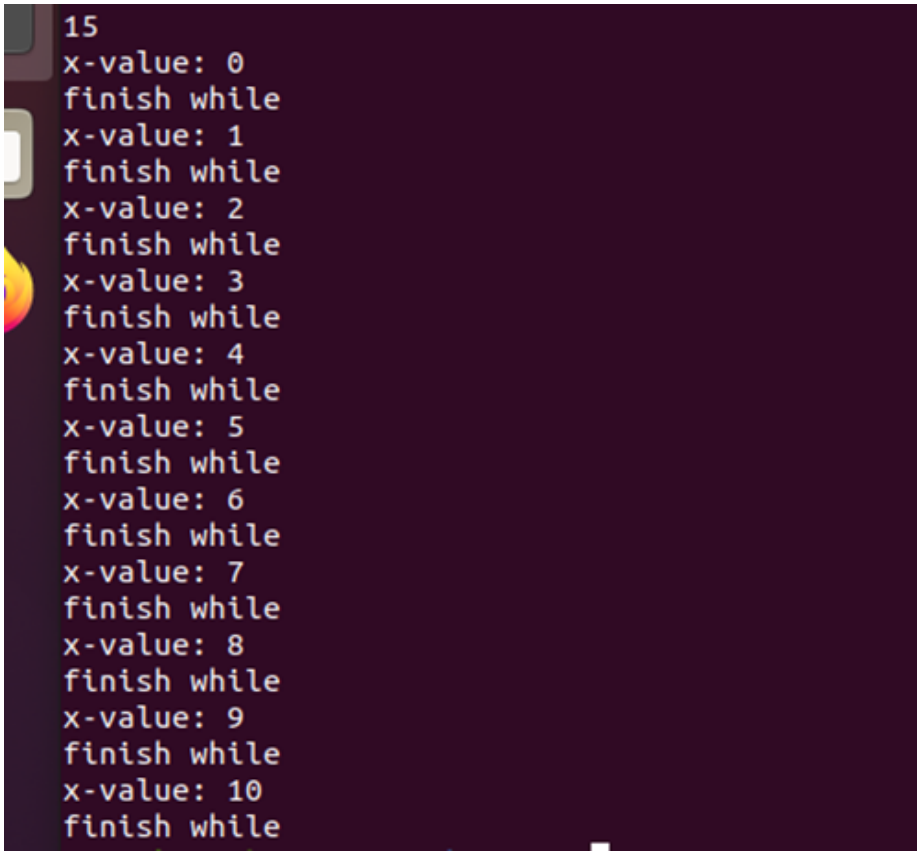
คำสั่งเรียกใช้ฟังก์ชันจากไฟล์ที่ชื่อว่า source.py

บทต่อไป เราจะทำการยกตัวอย่าง while/for loopของpython

สำหรับ while loop ทำการสร้างloop ในตัวอย่างเราจะทำ loop ที่ทั้งหมด 10 ครั้งโดยการตั้ง while(x<=10) แล้วxที่เราสร้างนั้นเริ่มต้นจาก 1

```
def count():
    x=0
    while(x<=10):
        print("x-value:",x)
        x=x+1
        print("finish while")
```

ตัวอย่างโค้ด



ผลลัพธ์ของโปรแกรม

ต่อสำหรับ For loopจะคล้ายๆกับ while loop แต่จะสั่นกว่าทำให้การเขียนนั้นง่ายมากยิ่งขึ้นโดยข้อเสียคือการสร้าง conditionในการนับนั้นจะไม่สามารถทำได้เท่ากับ while loop

ในตัวอย่างเราจะลองใช้ for loop โดยการนับกลับหลังจากrange 5-0 กลับหลังโดยพิม -1 ไปในcondition

```
        print('user incorrect')
def count():
    x=0
    while(x<=10):
        print("x-value:",x)
        x=x+1
        print("finish while")
def forloop():
    for a in range(5,0,-1):
        print("A-value:",a)
        a=a+1
        print("finish while")
```

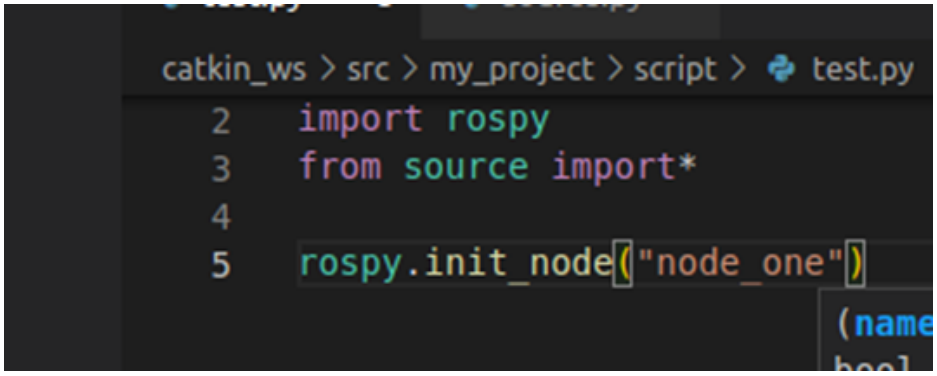
ตัวอย่างโค้ด

```
password correct
15
A-value: 5
finish while
A-value: 4
finish while
A-value: 3
finish while
A-value: 2
finish while
A-value: 1
finish while
oprocket@ubuntu:~/catkin_ws$
```

ผลลัพธ์ของโปรแกรม

ต่อไปเราจะเข้าสู่เนื้อหาของpython ที่เชื่อมโยงกับ ROS ขั้นแรกเราจะทำการสร้างnodeขึ้นมาด้วยคำสั่งของ rospy

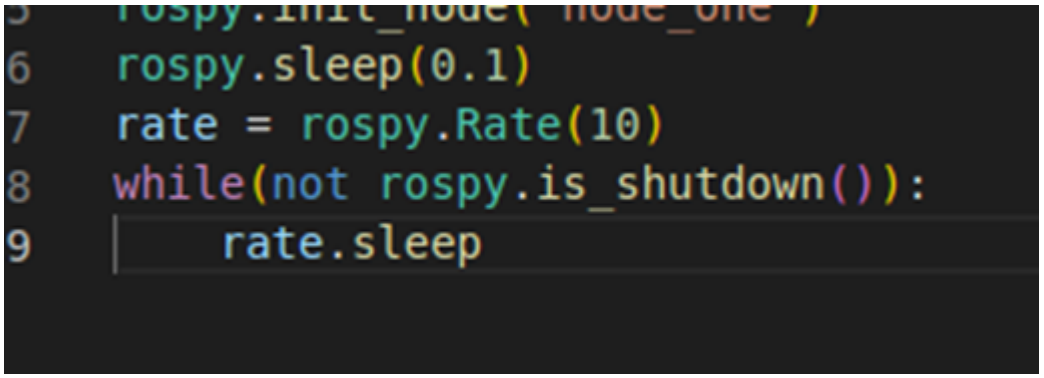
ตามตัวอย่างรูปด้านล่าง เราจะให้ชื่อว่า node_one.



การสร่าง node ของ ros

สำหรับการใช้ROSเราควรกำหนด rateของrosเพื่อไม่ให้ CPUของคอมพิวเตอร์ไม่ให้ทำงานหนักเกินไป

และเราทำwhile loopเพื่อที่จะเช็คว่าเครื่องนั้นถูกปิดอยู่หรือป่าว

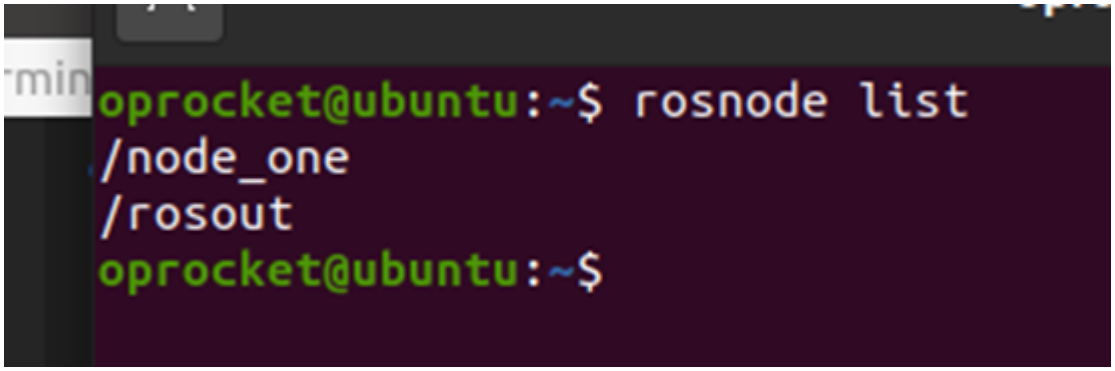


ตัวอย่างโค้ดเพื่อเช็คดูว่าเครื่องเปิดอยูหรือเปล่า

หลังจากที่เราสร้างnodeแล้วเราจะลองสั่ง roscore ดูว่า ที่เราทำนั้นสามารถเป็นตัว subscripe ได้หรือป่าว

เมื่อรันทั้ง roscore และตัว pythonที่เราสร้างแล้ว จะเช็คตัวnode ด้วยคำสั่ง rosnode list ในตัวของ terminal

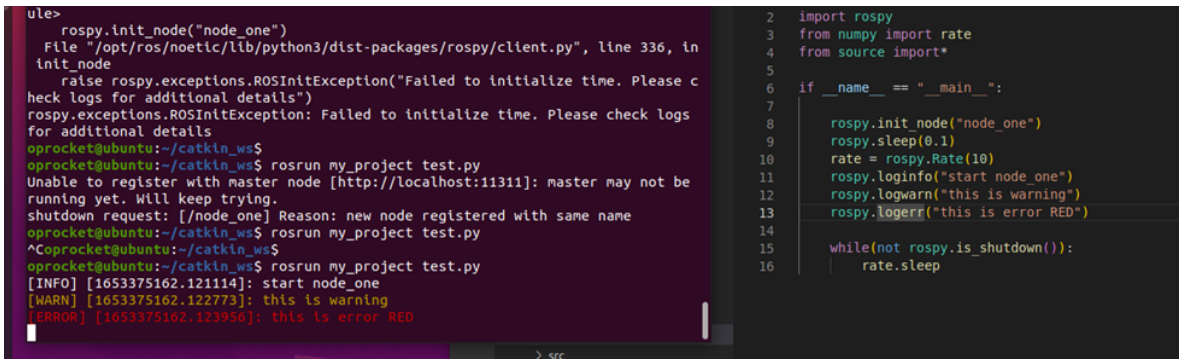
ตัวอย่างรูปด้านล่างจะเจอทั้ง 2 nodeถือเป็นอันถูกต้อง



ภาพรายชื่อ node

ต่อไปเป็นการทำเช็คดูว่าตัวอักษรของ ROS นั้นเป็นสีอะไรจาก ตัวของ info/warning/error จากการพิมพ์

คำสั่ง ของrospy.log/warn/err แล้วแต่จะคำสั่งจะมีสีของตัวอักษรนั้นๆตามรูปด้านล่าง



สีจะขึ้นอยู่กับคำสั่งที่เราใส่ไป

ต่อไปเราจะลองกับตัว tutlesim ที่คำสั่งในpython ขั้นแรกเราจะเขาไปเช็คว่าไฟล์ .py ของเรานั้นสามารถใช้ได้ไหมจากการใช้คำสั่ง ls ถ้าตัวไฟล์เป็นสีเขียวแปลว่าใช้ได้

```
procket@ubuntu:~/catkin_ws/src$ cd my_project
procket@ubuntu:~/catkin_ws/src/my_project$ cd script
procket@ubuntu:~/catkin_ws/src/my_project/script$ ls
__pycache__  source.py  test.py
procket@ubuntu:~/catkin_ws/src/my_project/script$
```

ภาพเช็คดูว่าไฟล์สามารถรันได้ไหม

หลังจากนั้นเราจะเขียนโค้ดpythonเพื่อควบคุมเต่าผ่านตัวของ ROS เริ่มจากการเขียนโค้ดpython ในไฟล์.pyที่เราสร้างขึ้นมา

เราก็จสร้างnodeขึ้นมาปกติ

package.xml | draw_circle.py ●

script > draw_circle.py

```python
#!/usr/bin/env python3
import rospy
from geometry_msgs import Twist

if __name__ == "__main__":
    rospy.init_node("draw circle")
    rospy.loginfo("node draw_circle is started")

```

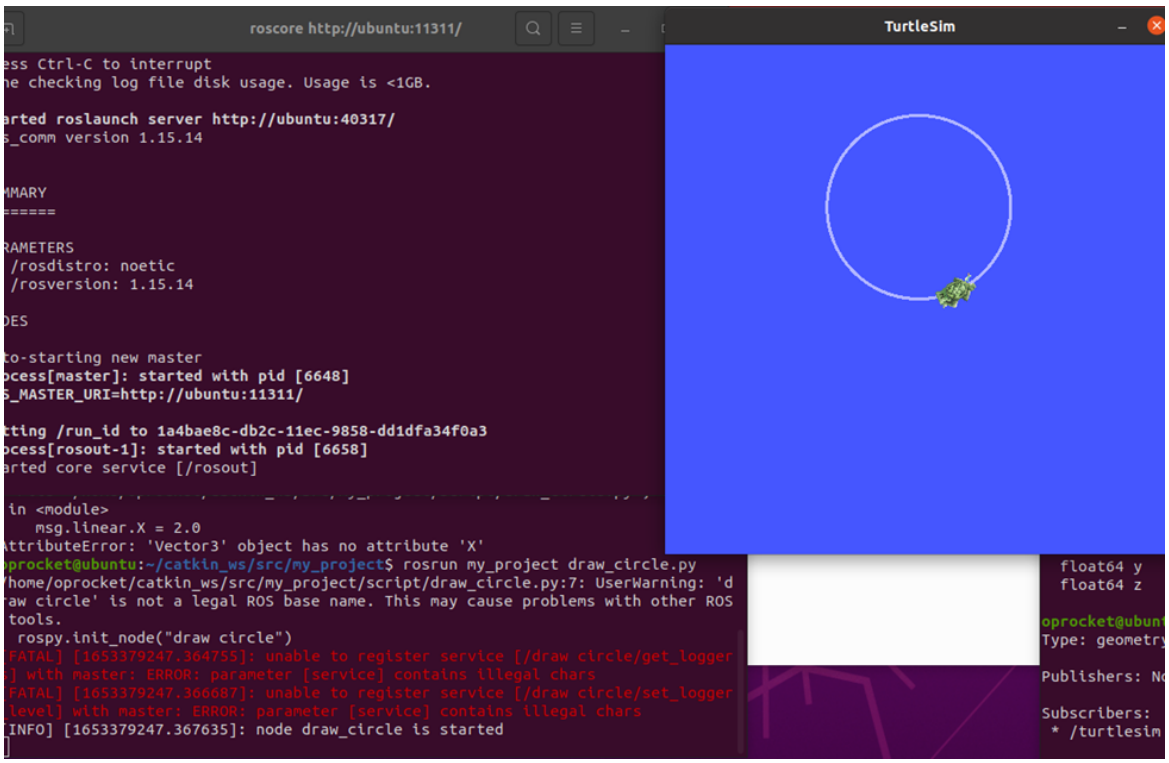
โค้ดตัวอย่างการสร้างnode

หลังจากนั้นเราทำให้nodeนี้เป็นpublisher ไว้คอยส่งค่าออกไปที่ subscriber จากการเขียนโค้ดต่อไปนี้

```python
import queue
import rospy
from geometry_msgs.msg import Twist

if __name__ == "__main__":
    rospy.init_node("draw circle")
    rospy.loginfo("node draw_circle is started")
    pub = rospy.Publisher("/turtle1/cmd_vel",Twist,queue_size=10)
    rate = rospy.Rate(2)

    while(not rospy.is_shutdown()):
        msg = Twist()
        msg.linear.x = 2.0
        msg.angular.z = 1.0
        pub.publish(msg)
```

โค้ดการสร้างnode ในการ Publsh ค่าออกไป

หลังจากนั้นเราก็เซฟไว้แล้วรันไฟล์.pyที่เราเขียนไว้ลงในterminal แล้ว หลังจากนั้นcodeที่เราเขียนก็จะ Publisher ไปสู่ turtle1/cmd_velเพื่อทำการเคลื่อนที่ตามโค้ดที่เขียน ดังรูปด้านล่าง

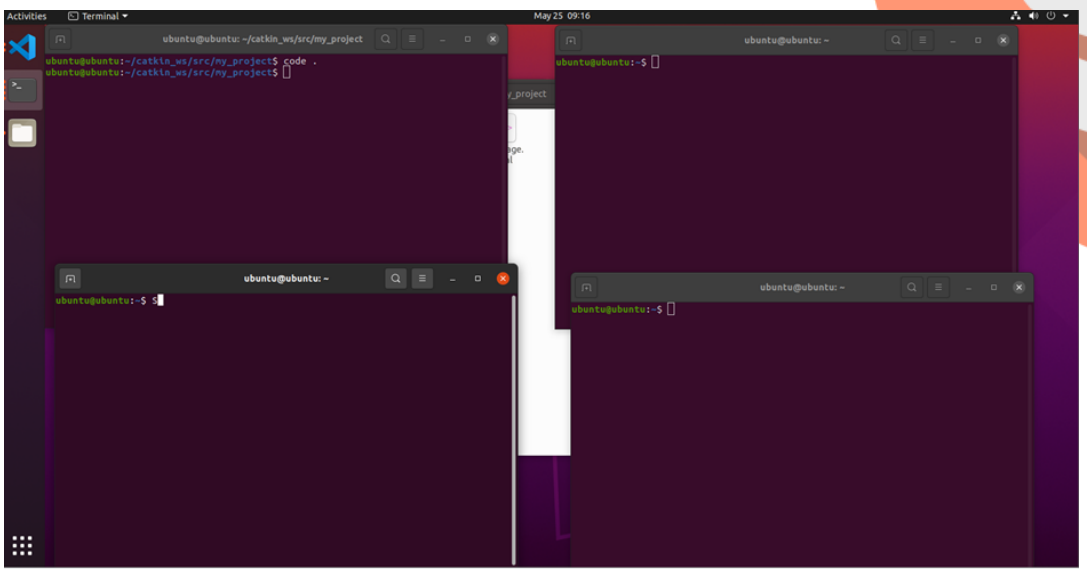


ผลลัพธ์ของโปรแกรม

บทที่3

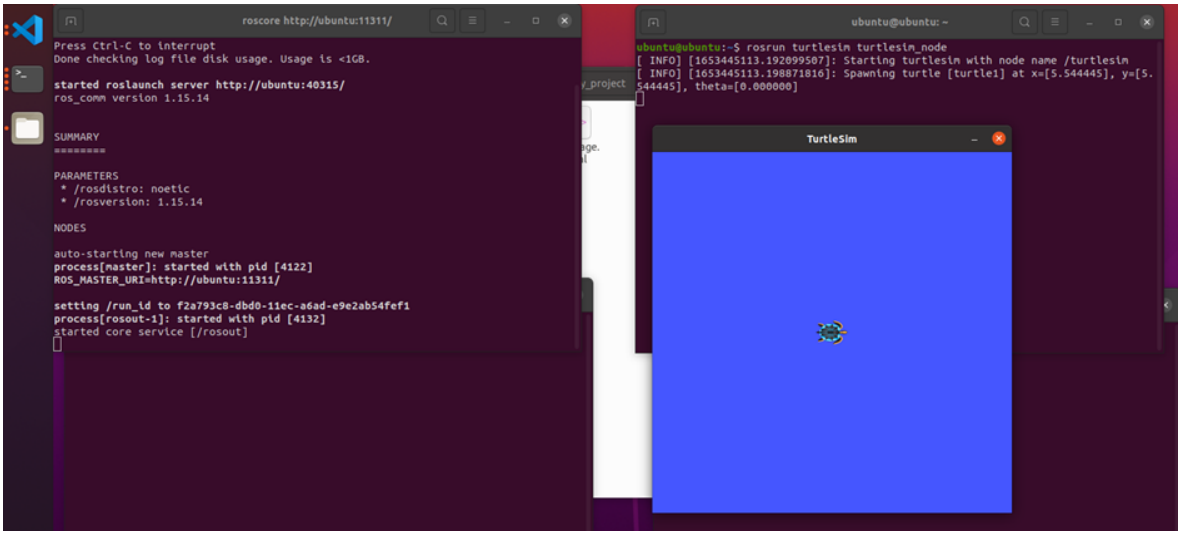
การทำ database cloud ลิ้งกับ turtlesim และการสร้าง GUI

สร้าง terminal ขึ้นมา 4 หน้าต่าง ให้ path ของ terminal ไปอยู่หน้า home ทั้งหมด



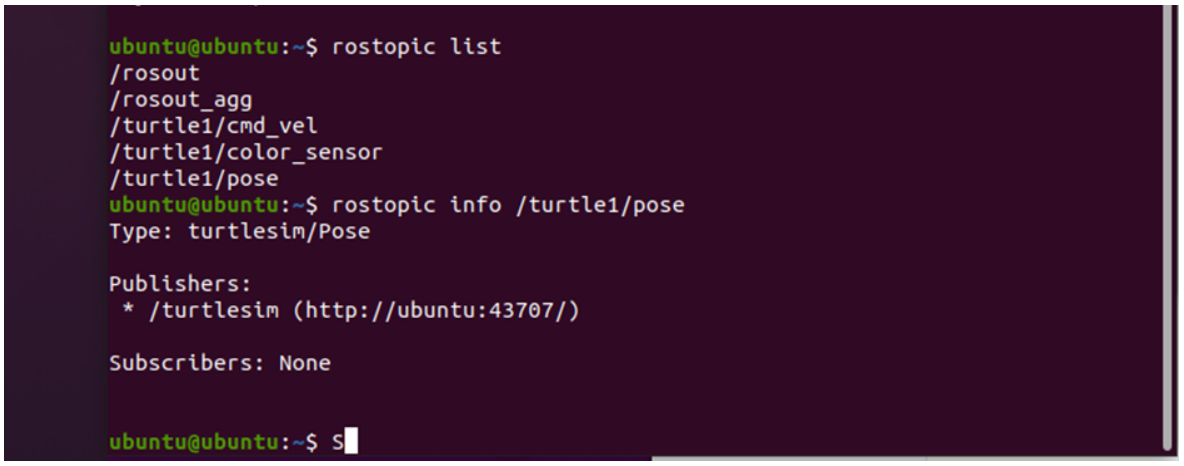
รูปการเปิด terminal ขึ้นมา 4 หน้าต่าง

Run คำสั่ง roscore อีกหน้าต่าง run คำสั่ง rosrun turtlesim turtlesim_node



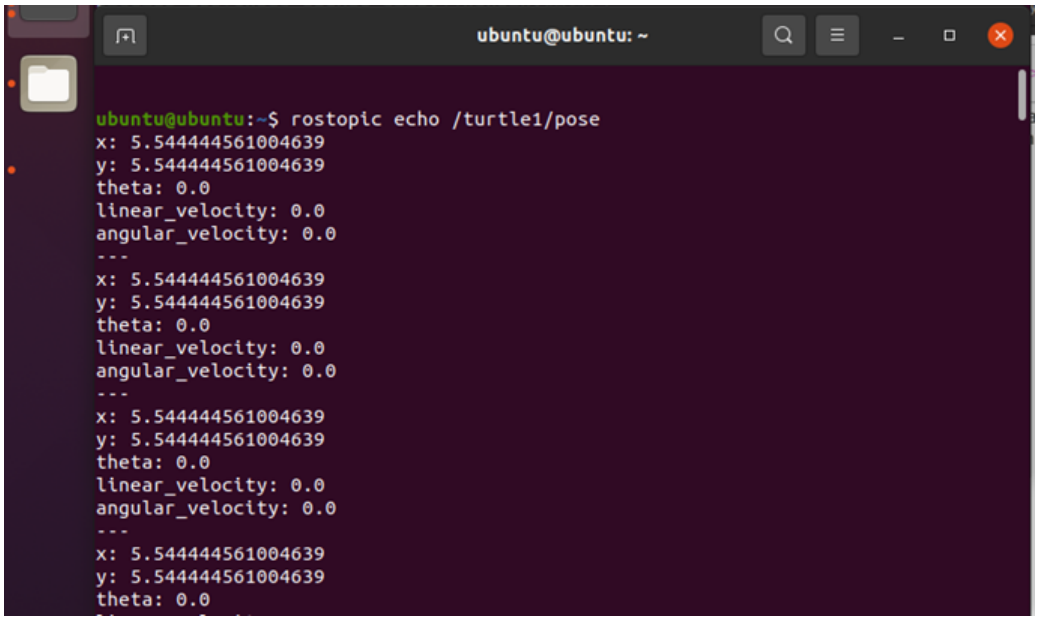
ผลลัพธ์เมื่อ run คำสั่ง rosrun turtlesim turtlesim_node

Run คำสั่ง rostopic เพื่อดู topic ของ turtlesim



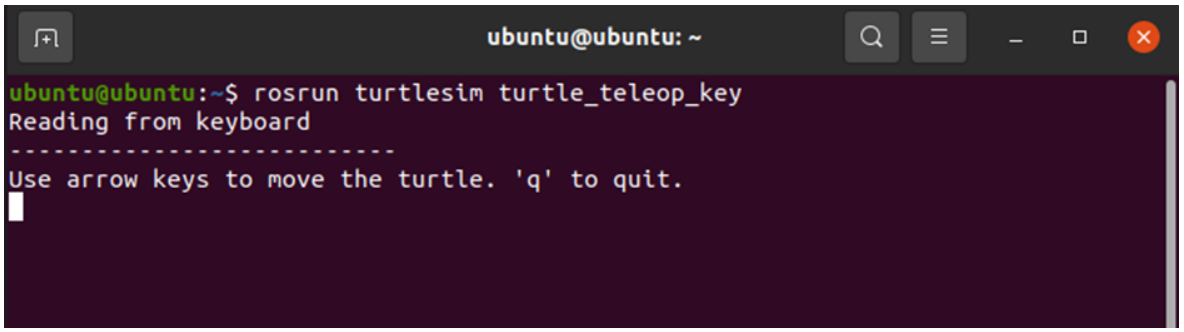
ภาพการ run คำสั่ง rostopic

Run คำสั่ง rostopic echo/turtle1/pose เพื่อดูว่ามีค่าอะไรบ้างที่ส่งมาจาก topic นี้



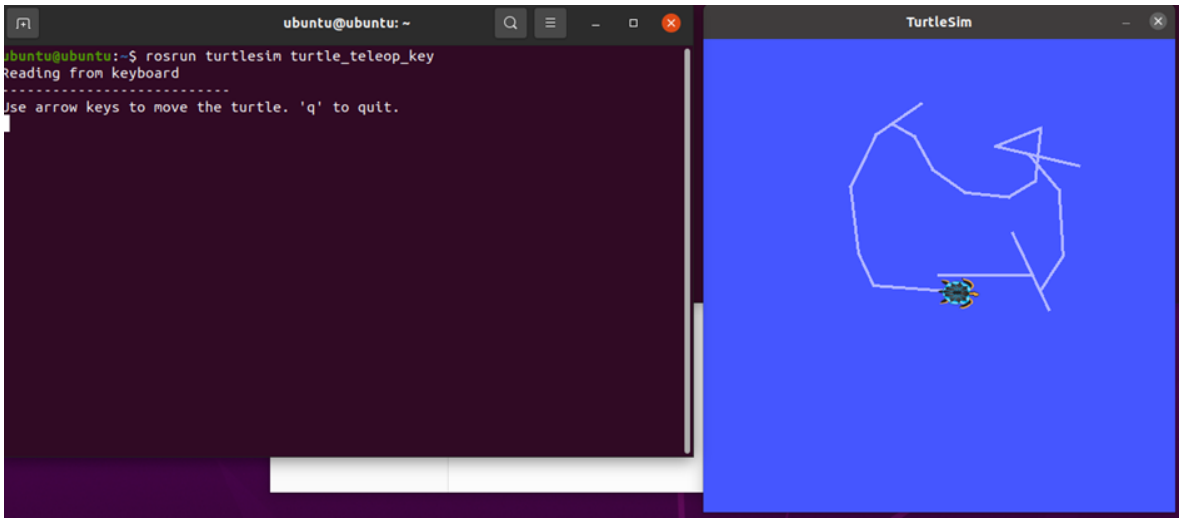
ผลัพธ์เมื่อ run ตำสัง rostopic echo/turtle1/pose

ขณะที่กำลัง run คำสั่ง rostopic echo/turtle1/pose อีก terminal ให้ run คำสั่ง rosrun turtlesim turtle_teleop_key



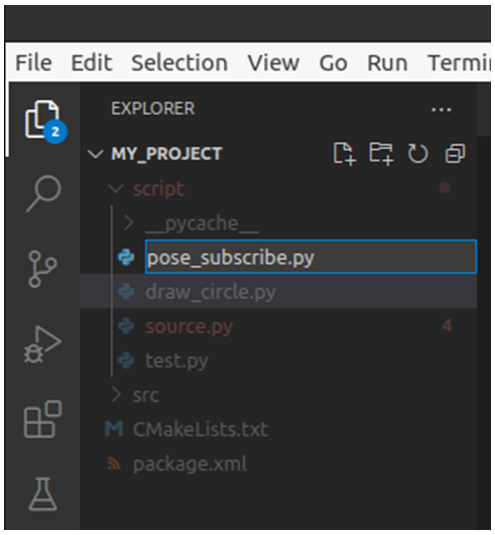
อีก terminal run คำสั่ง rosrun turltesim turtle_teleop_key

เมื่อขยับลูกศรบนkeyboard จะสามรถขยับทิศทางของเต่าในทิศทางที่ต้องการได้



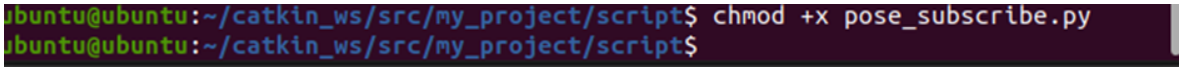
ผลลัพธ์เมื่อขยับและ run program

สร้างไฟล์ที่ชื่อ pose_subscribe.py ในไฟล์



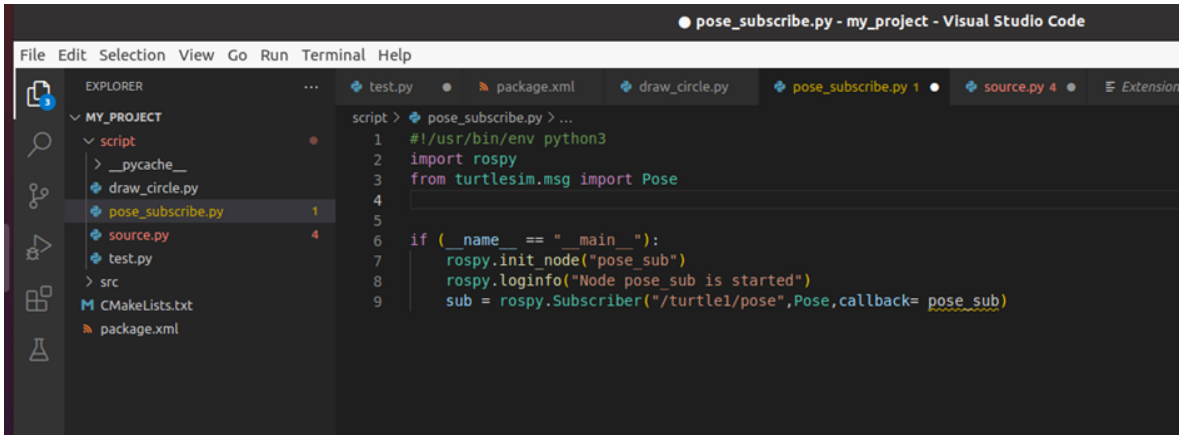
ภาพการสร้างไฟล์ที่ชื่อ pose_subscribe.py

ไปที่ ที่อยู่ของ ไฟล์ pose_subscribe.py เพื่อทำให้ไฟล์ executable ได้โดยใช้คำสั่ง chmod +x pose_subscribe.py



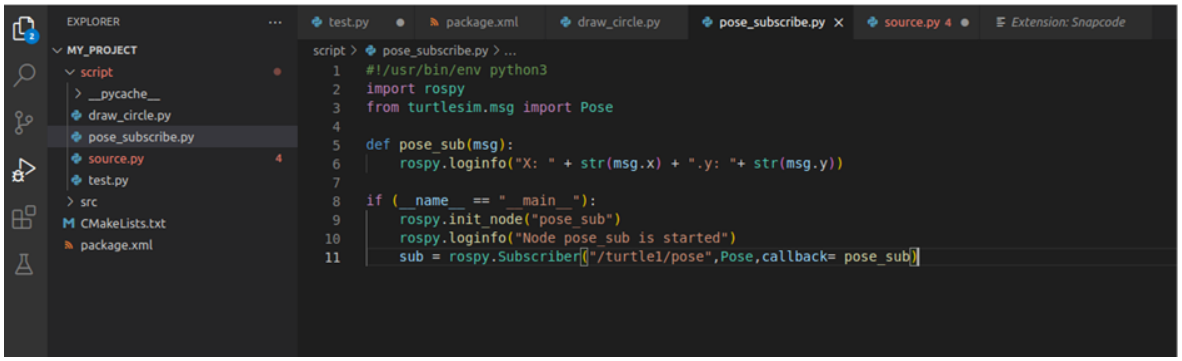
คำสั่งที่ทำให้ไฟล์ executable ได้

ทำการสร้าง node ที่เอาไว้ subsribe ข้อมูล จาก /turtle1/pose



โค้ดการสร้าง node เพื่อมารับค่าจาก node /turtle1/pose

ทำการสร้างfunction pose_sub เพื่อที่จะนำค่าที่ได้รับจาก callback=pose_sub ไปแสดงค่า



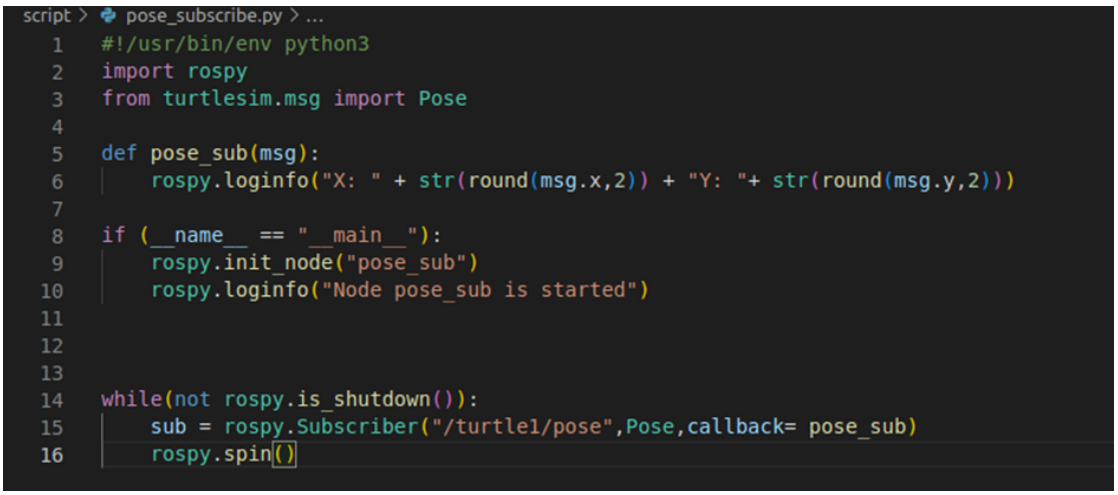
ภาพการสร้างฟังก์ชันขึ้นมาเพื่อรับค่าไปแสดงผลต่อ

Run โค้ดใน terminal เพื่อดูค่าที่ได้ subscribe มา

```
^Cubuntu@ubuntu:~$ rosrun my_project pose_subscribe.py
[INFO] [1653448000.869181]: Node pose_sub is started
[INFO] [1653448000.993809]: X: 4.754613876342773.y: 7.674404144287109
```

ผลลัพธ์ของโปรแกรม

เนื่องจากค่าที่ออกมาเป็นทศนิยมหลายตำแหน่งเราจึงใช้คำสั่ง round เพื่อปัดทศนิยมขึ้นเหลือ สองตำแหน่ง



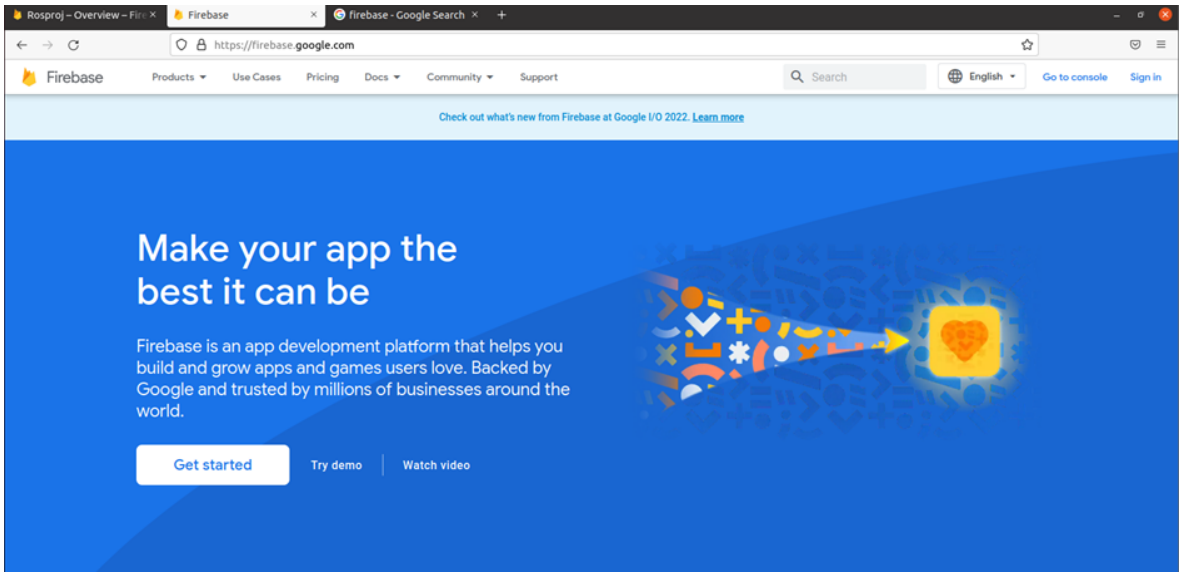
ภาพโค้ดใส่คำสั่ง round

เมื่อ run จะได้ค่าดังภาพข้างล่าง

```
ubuntu@ubuntu:~$ rosrun my_project pose_subscribe.py
[INFO] [1653449936.164320]: Node pose_sub is started
[INFO] [1653449936.176218]: X: 5.12Y: 5.51
[INFO] [1653449936.192194]: X: 5.12Y: 5.51
```

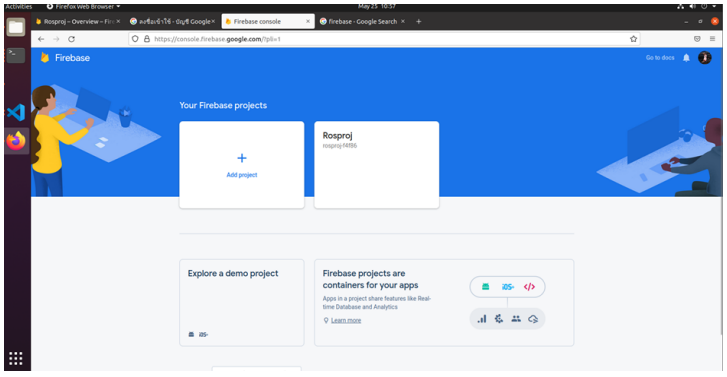
ผลลัพธ์เมื่อใส่คำสั่ง round

ทำการสมัคร firebase ใน ubuntu

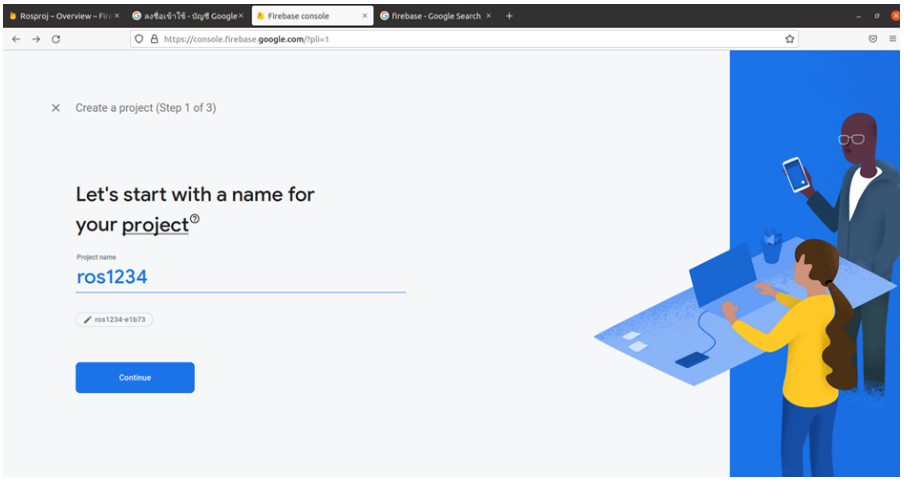


หน้าเว็บ firebase

เมื่อสมัครเรียบร้อยแล้วให้ กด get started และ สร้าง project ขึ้นมา

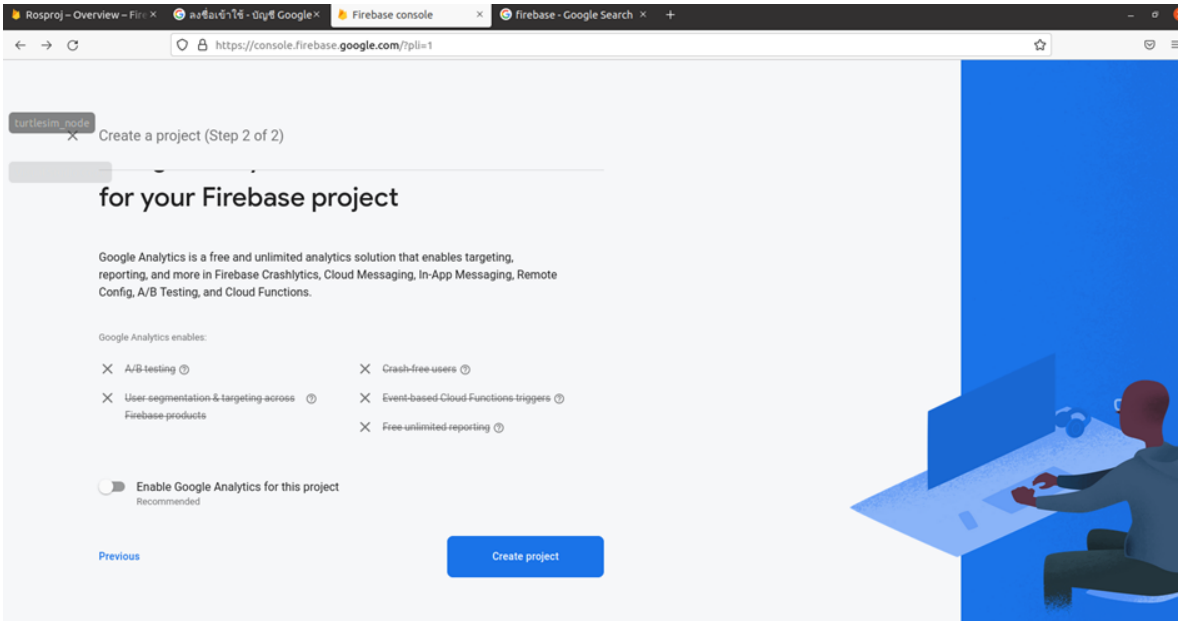


ตั้งชื่อ project

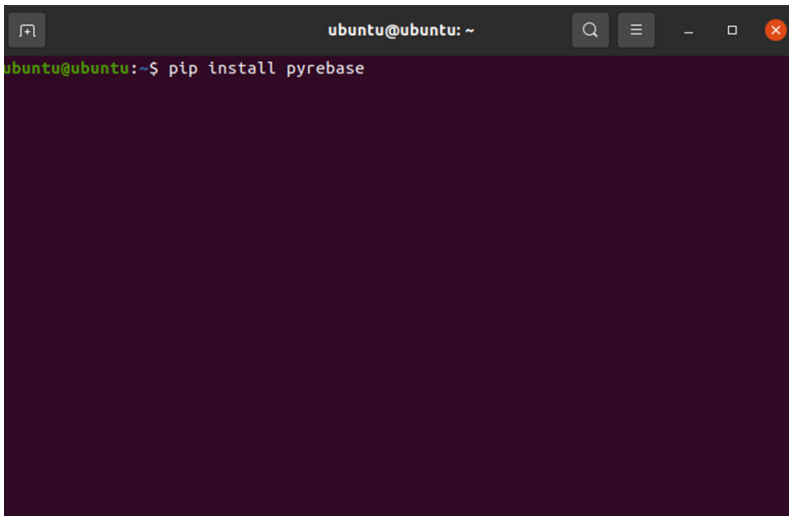


หน้าการตั้งชื่อ Project

กดยอมรับเงื่อนไข แต่ไม่กดใช้ google analytics และกด create project

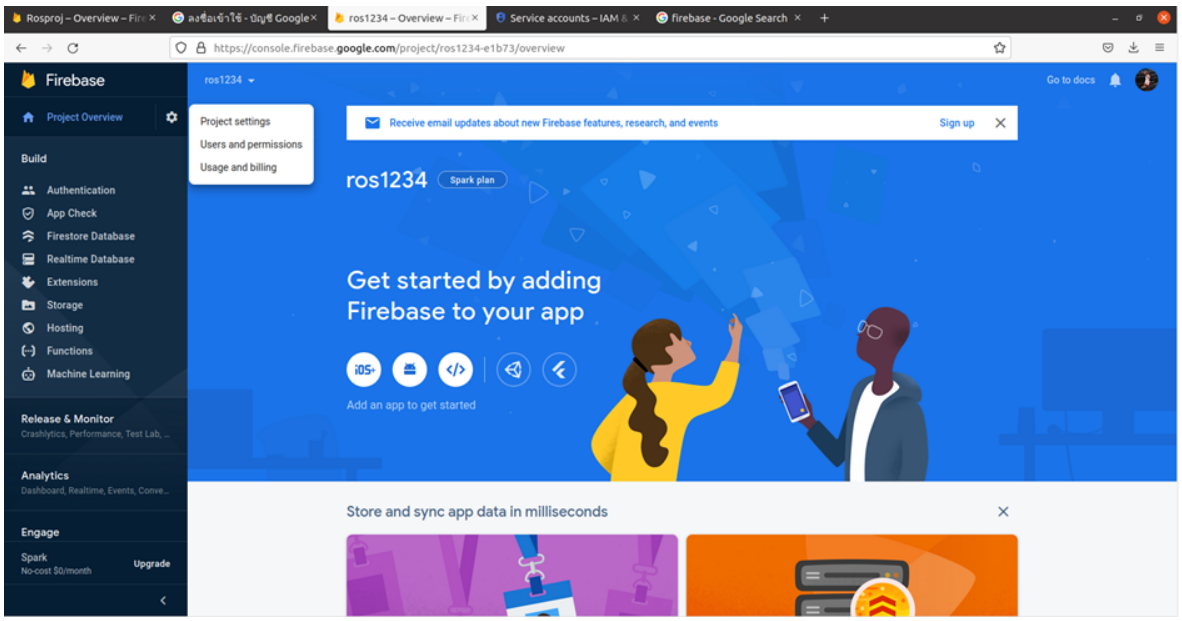


ทำการติดตั้ง library pyrebase เพื่อที่จะเชื่อมต่อกับ firebase ได้ โดยใช้คำสั่ง pip install pyrebaseในการติดตั้ง

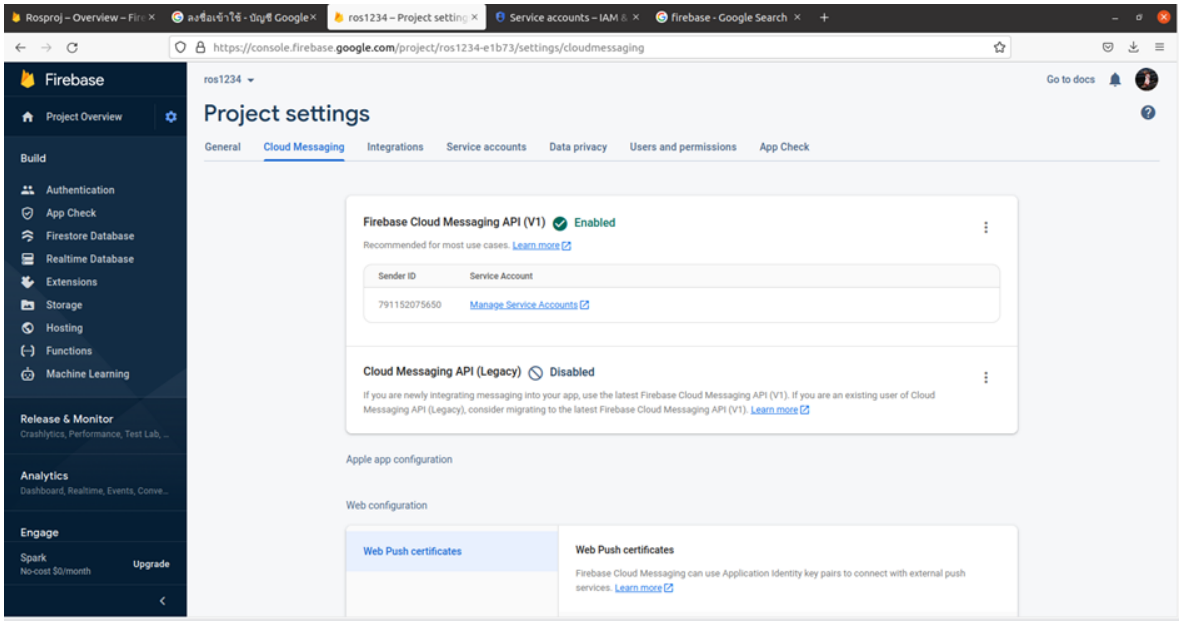


คำสั่งในการติดตั้ง pyrebase

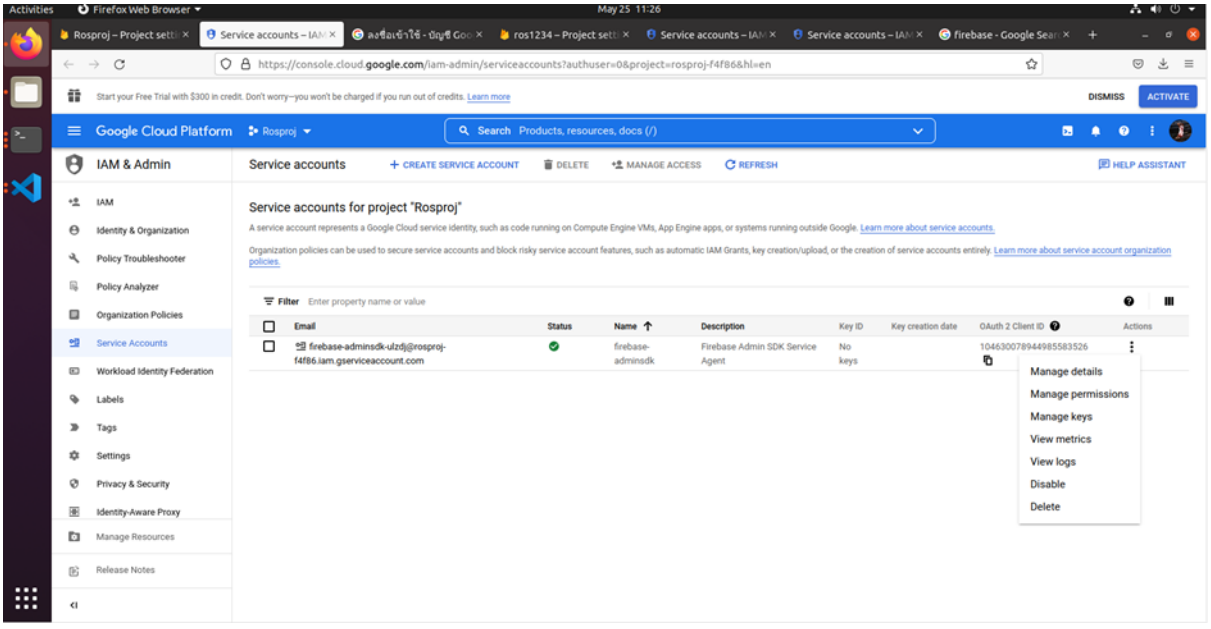
กดรูปฟันเฟืองด้านข้าง project overview และกด project setting



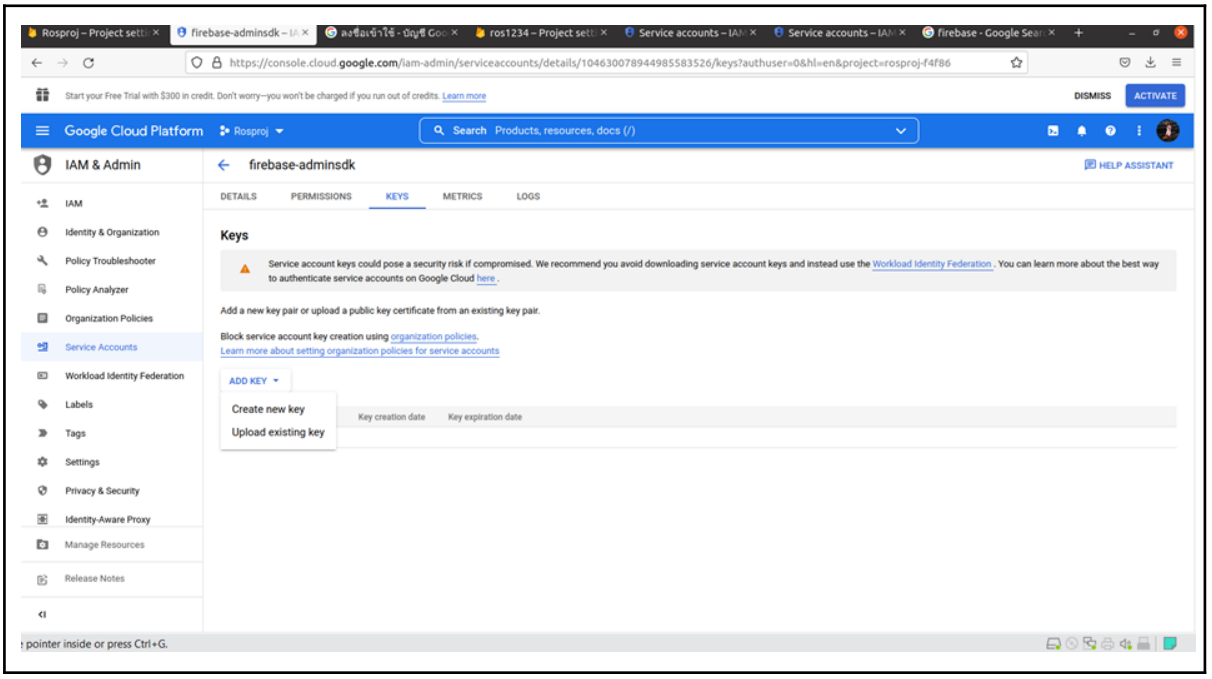
กดไปที่ clound messaging และกดที่ Manage Service Accounts



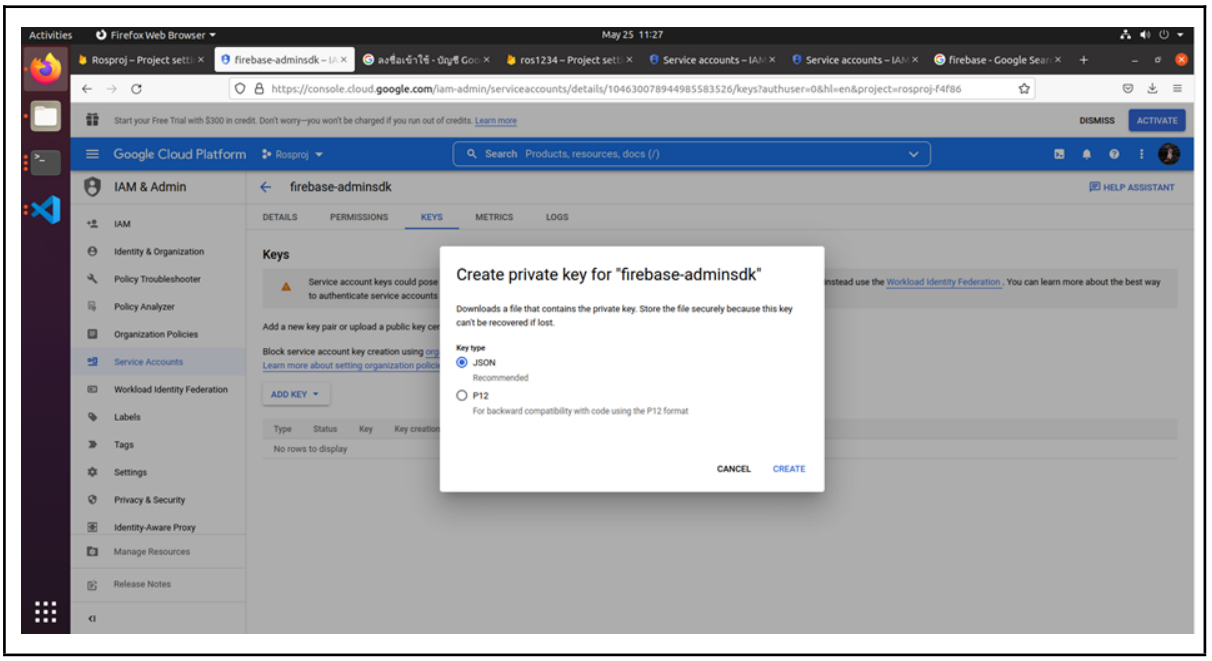
ให้กดจุดสามจุดที่ action และกด manage key



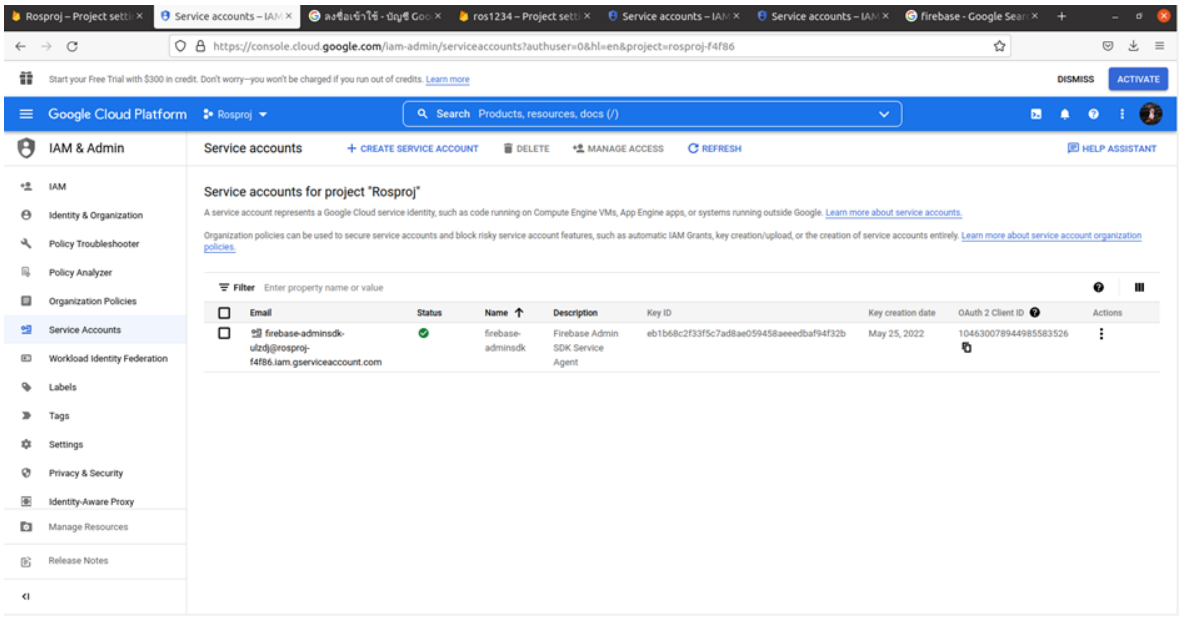
กด add key และ create new key



เลือก Json และ create

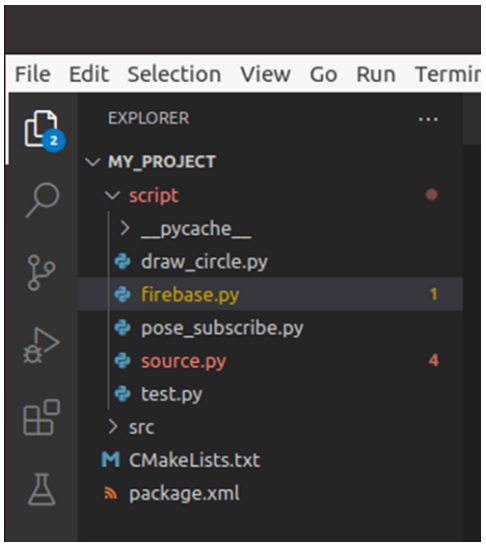


เมื่อกดสร้างkey เสร็จแล้ว กด service account จะเห็น key มา

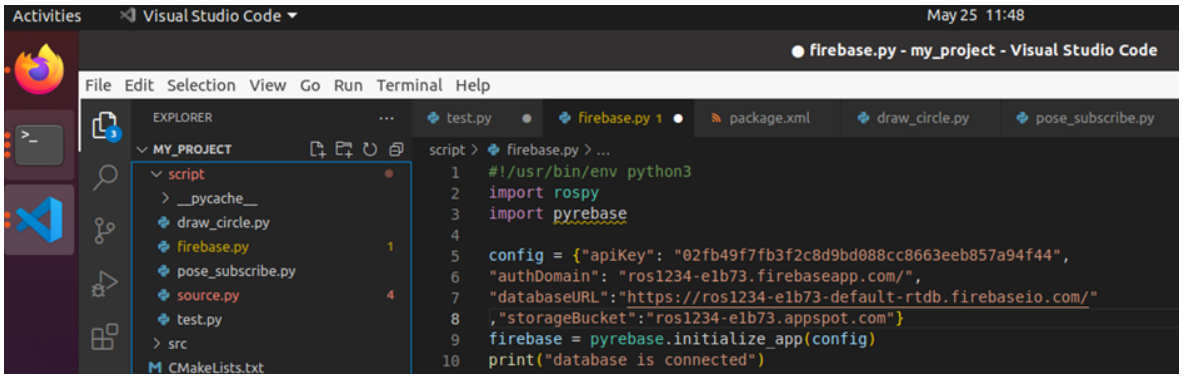


รูปkey

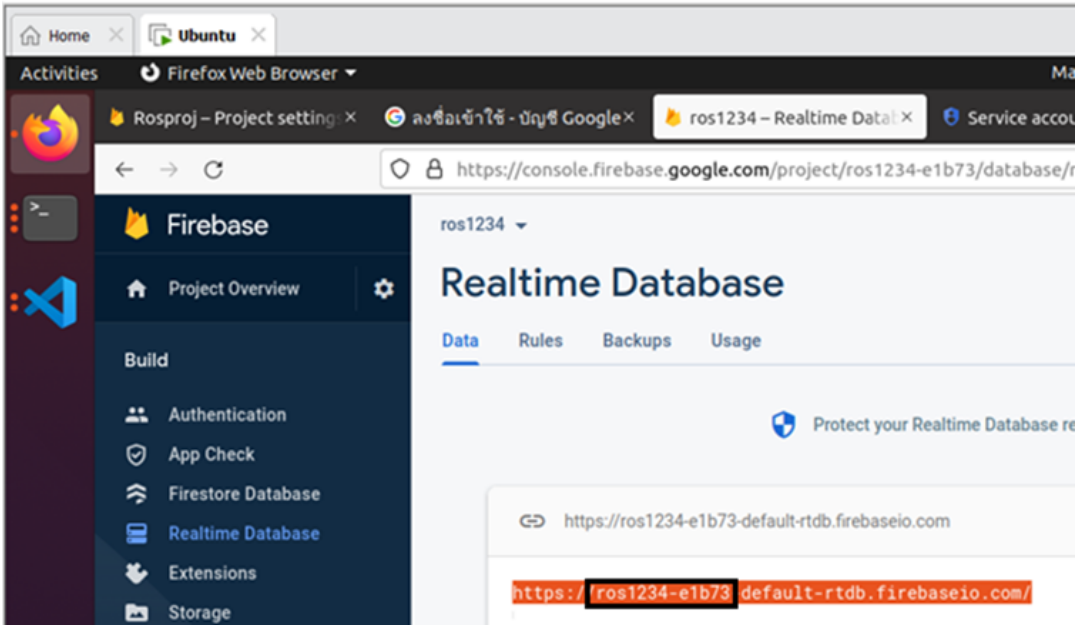
ให้เปิด vscode ขึ้นมาและสร้างไฟล์ firebase.py ขึ้นมาในFolder script



และเขียนโค้ด ดังภาพด้านล่าง และcopy key จาก service account มาใส่ที่ apikey



ในส่วนของ authDoamin ให้copy มาจาก Realtime databaseเฉพาะส่วนที่ ตีกรอบไว้



authDomain ของ แต่ละคน จะไม่เหมือนกัน

และเติม .firebaseapp.com/

```
"authDomain": "ros1234-e1b73.firebaseapp.com/",
```

ในส่วนdatabase url ให้ copy link ทั้งหมดมาใส่ได้เลย

```
"databaseURL":"https://ros1234-e1b73-default-rtdb.firebaseio.com/"
```

databaseURL

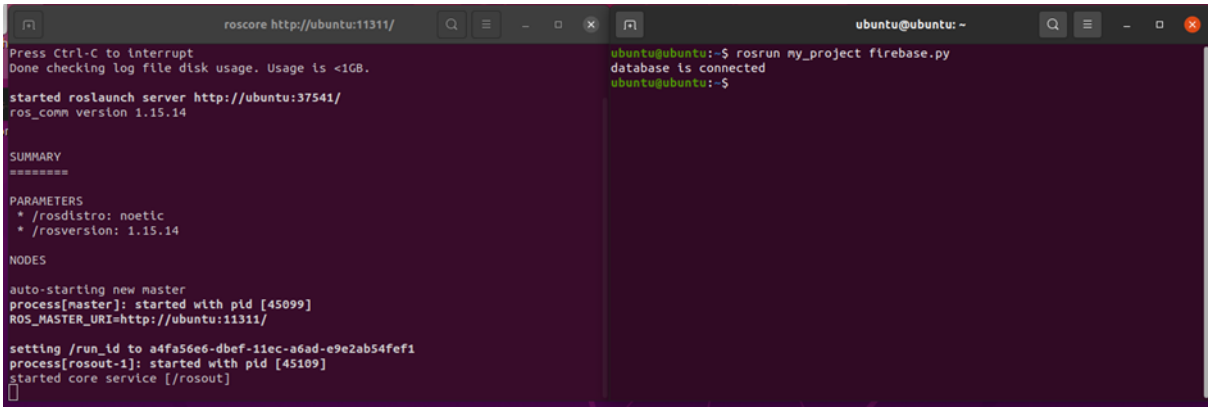
ส่วนสุดท้ายให้เติม .appspot.com ดังภาพ

```
,"storageBucket":"ros1234-e1b73.appspot.com"}
```

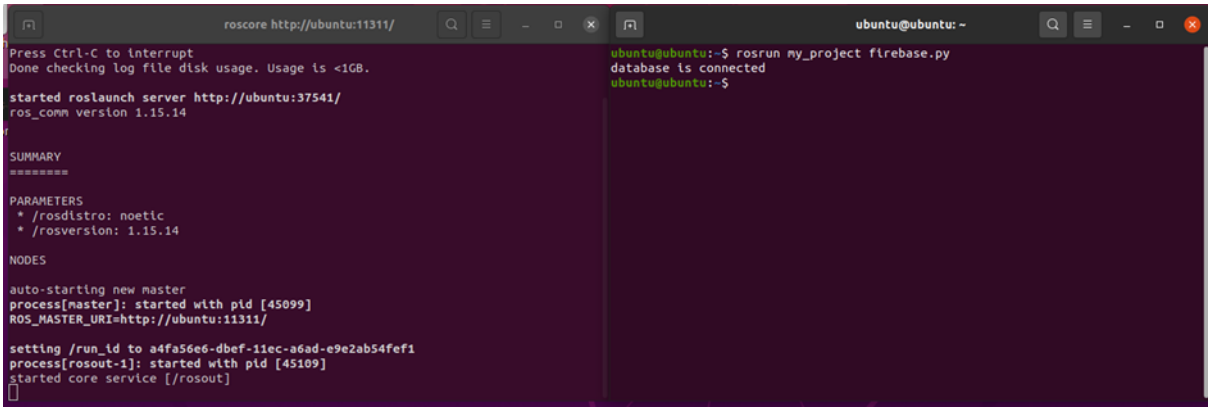
จากนั้นจะได้โค้ดดังภาพ

```
#!/usr/bin/env python3
import rospy
import pyrebase

config = {"apiKey":
"02fb49f7fb3f2c8d9bd088cc8663eeb857a94f44",
"authDomain": "ros1234-e1b73.firebaseapp.com/",
"databaseURL":
"https://ros1234-e1b73-default-rtdb.firebaseio.com/"
,"storageBucket":"ros1234-e1b73.appspot.com"}
firebase = pyrebase.initialize_app(config)
print("database is connected")

db = firebase.database()

db.child("a").update({"data" : 456})
```

เปิดterminal ขึ้นมาสอง หน้าต่าง หน้าต่างซ้ายพิมพ์คำสั่ง roscore อีกหน้าต่าง พิมพ์คำสั่ง run ไฟล์ firebase.py



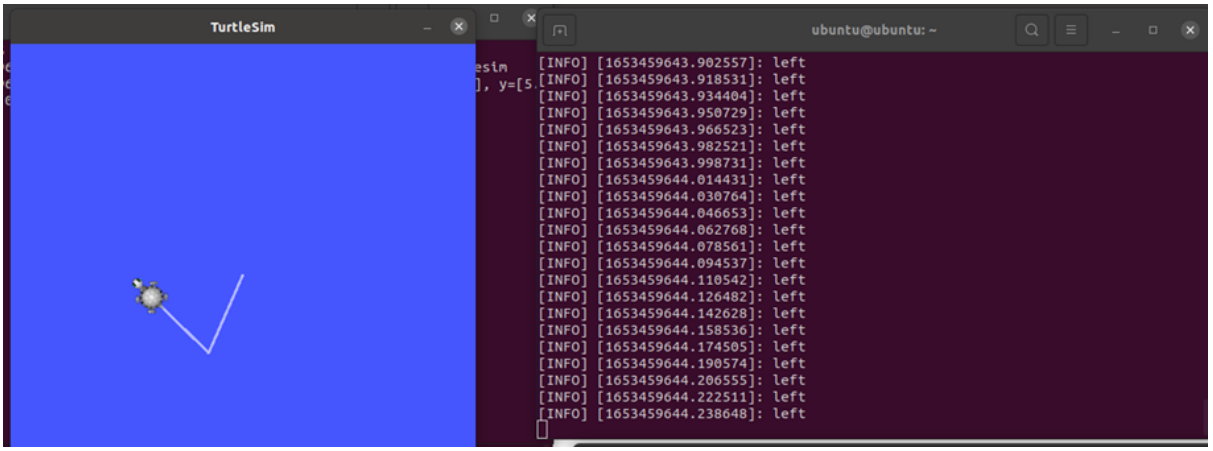
จากนั้นให้ไปที่firebase เพื่อตรวจสอบว่าข้อมูลเราได้ถูกส่งไปยัง firebase หรือยัง หากถูกต้องจะขึ้นดังภาพด้านล่าง



ทำการสร้างไฟล์ขึ้นมา1 ไฟล์ ชื่อไฟล์ decision.py จะเป็นการบอกตำแหน่งของเต่า

```python
#!/usr/bin/env python3
from matplotlib.pyplot import tick_params
import rospy
import pyrebase
from turtlesim.msg import Pose

config = {"apiKey": "02fb49f7fb3f2c8d9bd088cc8663eeb857a94f44",
"authDomain": "ros1234-e1b73.firebaseapp.com/",
"databaseURL":"https://ros1234-e1b73-default-rtdb.firebaseio.com/"
,"storageBucket":"ros1234-e1b73.appspot.com"}
firebase = pyrebase.initialize_app(config)
print("database is connected")
db = firebase.database()

def pose_sub(msg):
    if(msg.x > 0 and msg.x<= 3.6):
            rospy.loginfo("left")
            db.child("a").update({"data" : "left"})
    if(msg.x> 3.6 and msg.x <=7.2):
            rospy.loginfo("centre")
            db.child("a").update({"data" : "centre"})
    if(msg.x > 7.2 and msg.x <= 11):
            rospy.loginfo("right")
            db.child("a").update({"data" : "right"})


if (__name__ == "__main__"):
    rospy.init_node("pose_sub")
    rospy.loginfo("Node pose_sub is started")


while(not rospy.is_shutdown()):
    sub = rospy.Subscriber("/turtle1/pose",Pose,callback= pose_sub)
    rospy.spin()
```

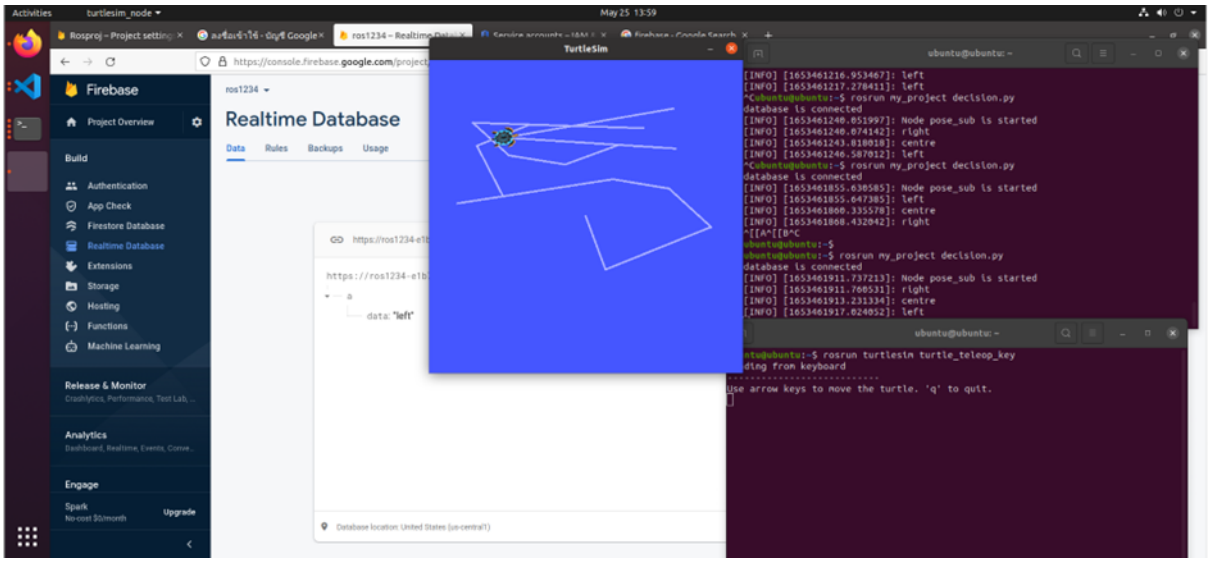
ภาพนี้เต่าอยู่ทางฝั่งซ้ายขอจอ ตัวterminalด้านขวาได้บอกตำแหน่งว่าอยู่ด้านซ้าย



เราสามาถตรวจสอบว่าค่าตำแหน่งของเต่าได้ทำการ upload ไปบน firebase หรือเปล่า แต่เมื่อเปลี่ยนตำแหน่งของเต่าแล้วแต่ค่าที่แสดงมาจาก termimal ยังไม่เปลี่ยนแปลงเพราะว่าเราส่งค่าขึ้นไปเยอะเกินไป จึงต้องมีการเปลี่ยนแปลงโค้ด

```python
#!/usr/bin/env python3
from matplotlib.pyplot import tick_params
import rospy
import pyrebase
from turtlesim.msg import Pose

config = {"apiKey": "02fb49f7fb3f2c8d9bd088cc8663eeb857a94f44",
"authDomain": "ros1234-e1b73.firebaseapp.com/",
"databaseURL":"https://ros1234-e1b73-default-rtdb.firebaseio.com/"
,"storageBucket":"ros1234-e1b73.appspot.com"}
firebase = pyrebase.initialize_app(config)
print("database is connected")
db = firebase.database()

def pose_sub(msg):
    if(msg.x > 0 and msg.x<= 3.6):
            rospy.loginfo("left")
            db.child("a").update({"data" : "left"})
    if(msg.x> 3.6 and msg.x <=7.2):
            rospy.loginfo("centre")
            db.child("a").update({"data" : "centre"})
    if(msg.x > 7.2 and msg.x <= 11):
            rospy.loginfo("right")
            db.child("a").update({"data" : "right"})


if (__name__ == "__main__"):
    rospy.init_node("pose_sub")
    rospy.loginfo("Node pose_sub is started")


while(not rospy.is_shutdown()):
    sub = rospy.Subscriber("/turtle1/pose",Pose,callback= pose_sub)
    rospy.spin()

```

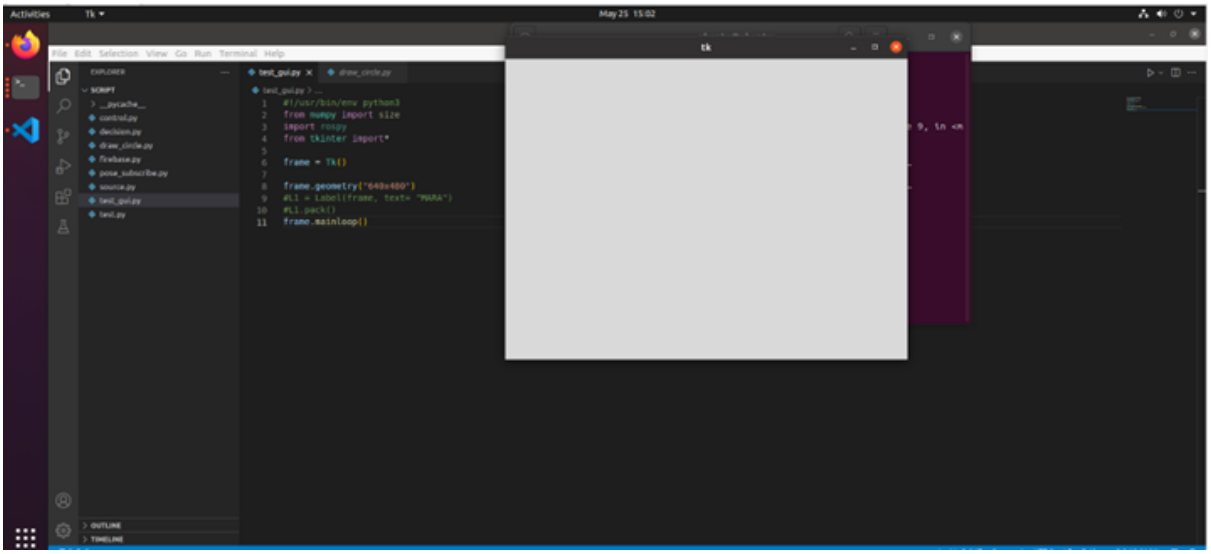
เมื่อ run โปรแกรมจะได้หน้าต่างแบบนี้



ในส่วนนี้เราจะทำการสร้างGUI ขึ้นมาโดยใช้ library import tkinter เราจะสร้างหน้าต่างขนาด 640x480

```python
#!/usr/bin/env python3
from numpy import size
import rospy
from tkinter import*

frame = Tk()

frame.geometry("640x480")
L1 = Label(frame, text= "MARA")
L1.pack()
frame.mainloop()


```

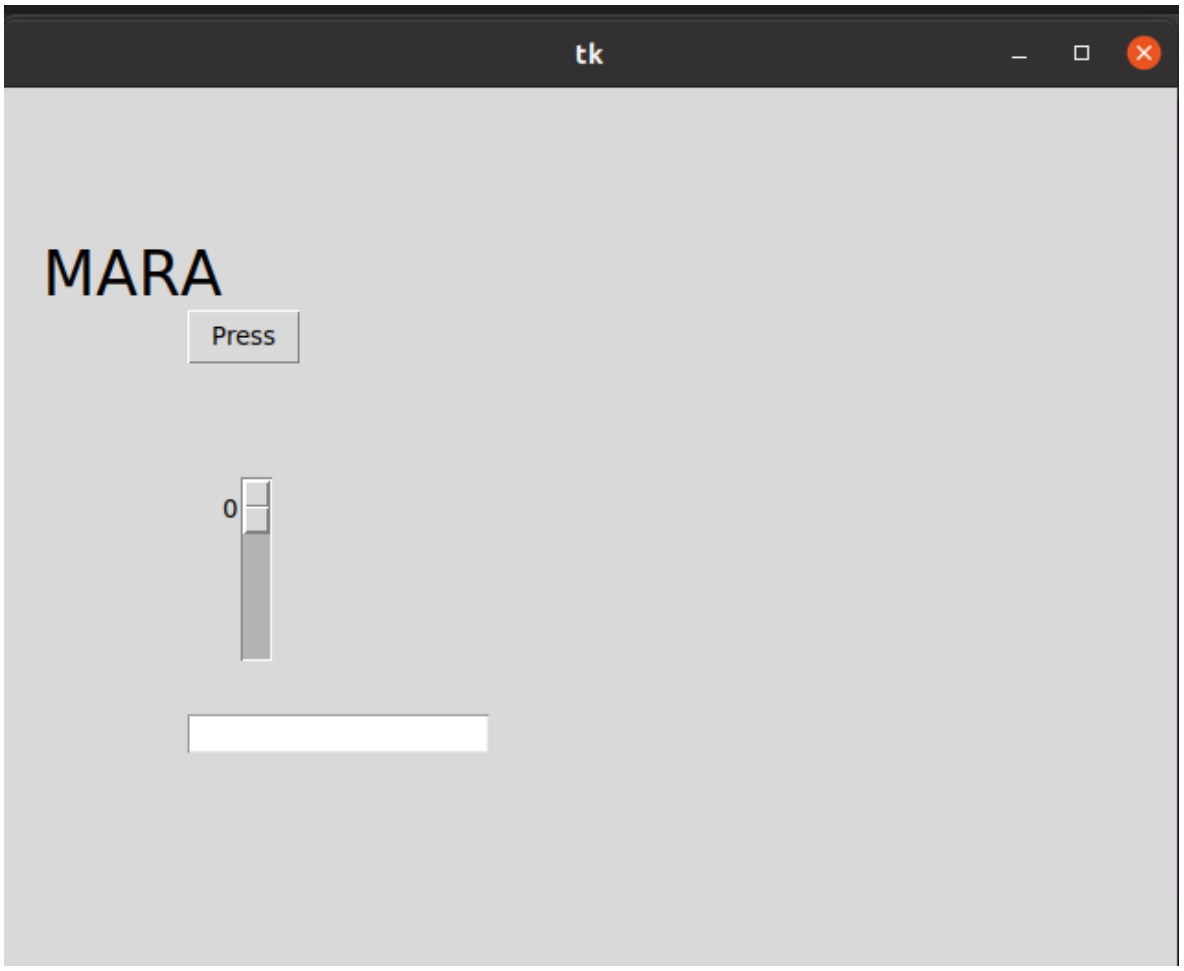
เมื่อrun โปรแกรมจะได้หน้าต่างแบบนี้



ในขั้นนี้เราได้เพิ่ม ตัวหนังสือ ปุ่ม แถบเลื่อน และช่องกรอกข้อมูล

```python
#!/usr/bin/env python3
from numpy import size
import rospy
from tkinter import*

frame = Tk()

frame.geometry("640x480")
L1 = Label(frame, text= "MARA", font= ('Aerail',25))
L1.place(x=20,y=80)
B = Button(frame, text = "Press")
B.place(x=100, y= 120)
S = Scale(frame)
S.place(x=100, y=210)
E = Entry(frame)
E.place(x=100,y=340)

frame.mainloop()
```

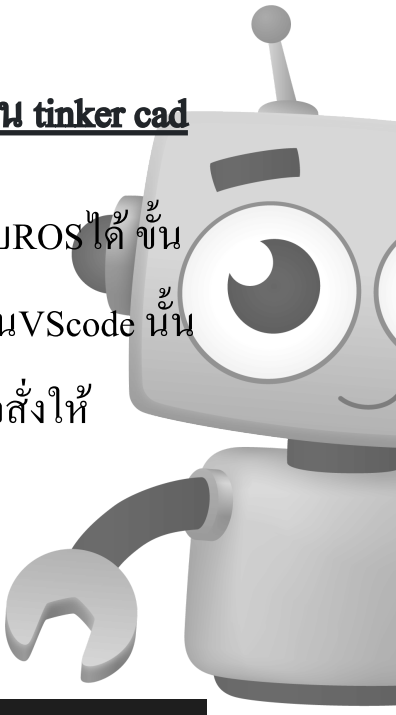
จะได้ผลลัพธ์ดังรูป



ผลลัพธ์ของโค้ด

บทที่4

ใช้ arduino สื่อสารกับระบบ ROS และก็มีการสร้าง simulation ใน tinker cad



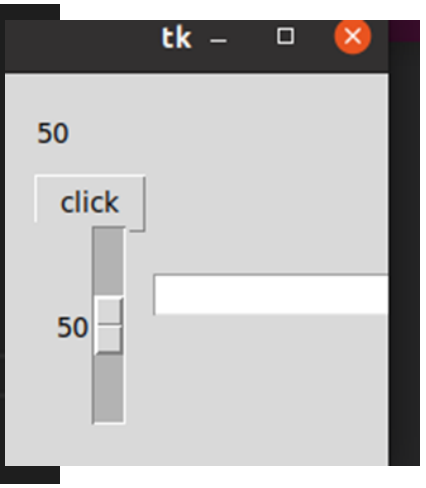
ต่อไปเราจะเข้าสู่ขั้นตอนของการสร้างGUIที่สามารถ activeกับROSได้ ขั้นแรกเราจะลองทำให้function ของbutton จากกานเขียนโค้ด Python ในVScode นั้นสามารถส่งcommand ด้วยคำสั่งโค้ดรูปด้านล่าง เมื่อcommandไปแล้วสั่งให้commandนั้นออกคำสั่งอะไรบ้างตามที่เราได้เขียนลงไป

ตัวอย่างรูปด้านล่าง

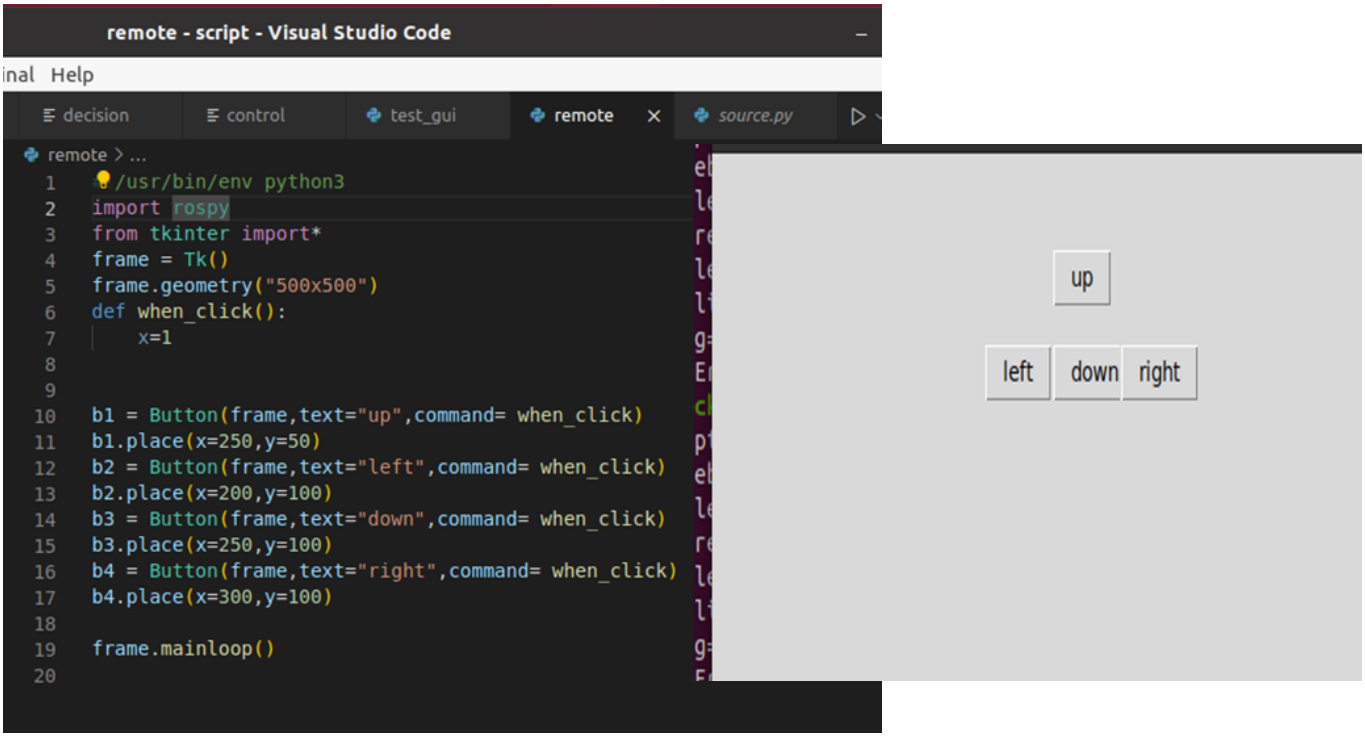
```
frame.geometry("200x200")

def when_click():
    L1.config(text="hi")

L1 = Label(frame,text="hello")
L1.place(x=20,y=20)
b1 = Button(frame,text="click",command= when_click)
```

ต่อไปจะลองใช้ของตัว scale ดูว่าจะสามารถเล่นกับค่าของ scaleได้ไหมโดยการเราจะให้เมื่อเลื่อน scale แล้วจะไปคำสั่งของ Show text ตามรูปตัวอย่างด้านล่าง

```
def when_click():
    v=s.get()
    L1.config(text=str(v))

L1 = Label(frame,text="hello")
L1.place(x=20,y=20)
b1 = Button(frame,text="click",command= when_click)
b1.place(x=20,y=50)
s = Scale(frame)
s.place(x=20,y=75)
```



เมื่อเราเรียนรู้วิธีการเราการทำ command ของแต่ละปุ่มได้แล้วทีนี้ขั้นตอนต่อไปเราจะมาทำการลองเชื่อโยงกับ Ros โดยตัวอย่างนี้เราจะลองสร้างปุ่มขึ้นมาเพื่อบังคับการเดินของตัวเต่าใน turtlesim ดูว่าจะได้หรือไม่ โดยขั้นแรกเราจะสร้างปุ่มขึ้นมาก่อนจากโค้ดที่เราเคยสร้างขึ้นมา ตัวอย่างโค้ดด้านล่าง



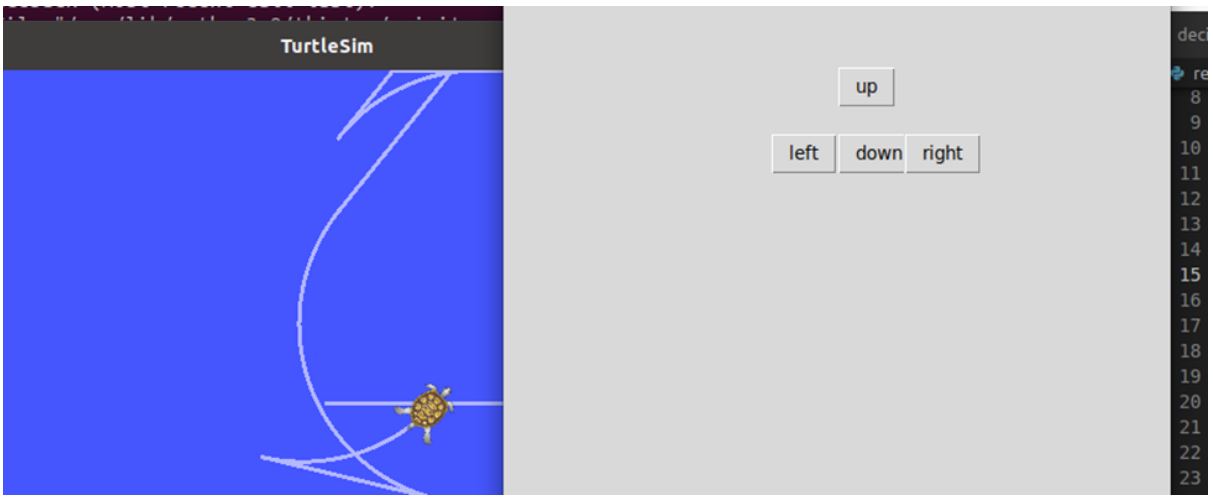
ต่อไปเราจะทำให้ค่าของการกดปุ่มนี้ส่งไปให้ turtlesim ของ function cmd_vel โดยจะส่งในรูปของ Twist ดังนั้นเราจะ import Twist นั้นขึ้นมาก่อนแล้วสร้าง node และ publisher ขึ้นมาตามปกติแบบที่เราเคยสร้างมาจากบทก่อนหน้านี้

จากนั้นเราจะทำการกำหนดการส่งค่าจากปุ่มไปเป็นเดินหน้าถอยหลังซ้ายขวาตามปกติที่เราเคยทำไปโดยค่าความเร็วหรือมุมนั้นเราสามารถสร้างกำหนดขึ้นมาเองได้แล้วแต่ที่เราต้องการ โค้ดตัวอย่างตามด้านล่าง

```
def FW():
    cmd = Twist()
    cmd.linear.x=3.0
    cmd.angular.z=0
    pub.publish(cmd)
def L():
    cmd = Twist()
    cmd.linear.x=3.0
    cmd.angular.z=1
    pub.publish(cmd)
def D():
    cmd = Twist()
    cmd.linear.x=-3.0
    cmd.angular.z=0
    pub.publish(cmd)
def R():
    cmd = Twist()
    cmd.linear.x=3.0
    cmd.angular.z=-1
    pub.publish(cmd)
```

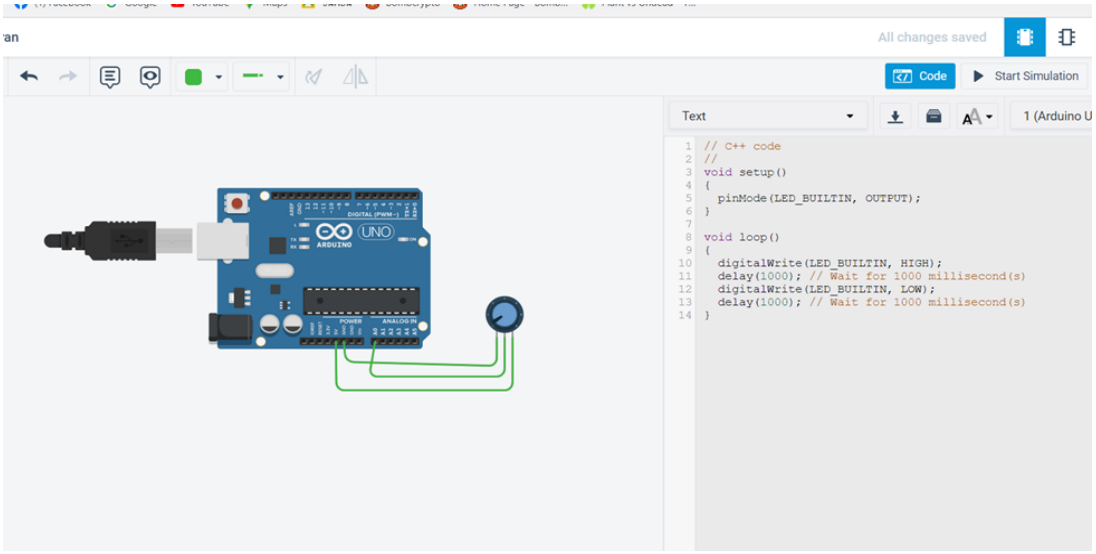
```
b1 = Button(frame,text="up",command= FW)
b1.place(x=250,y=50)
b2 = Button(frame,text="left",command= L)
b2.place(x=200,y=100)
b3 = Button(frame,text="down",command= D)
b3.place(x=250,y=100)
b4 = Button(frame,text="right",command= R)
b4.place(x=300,y=100)
```

เราก็จะได้ค่าของปุ่มที่เราสามารถส่งค่าคำสั่ง ROS ไปสู่ turtlesim ได้ เป็นการจบตัวของเรื่อง GUI.

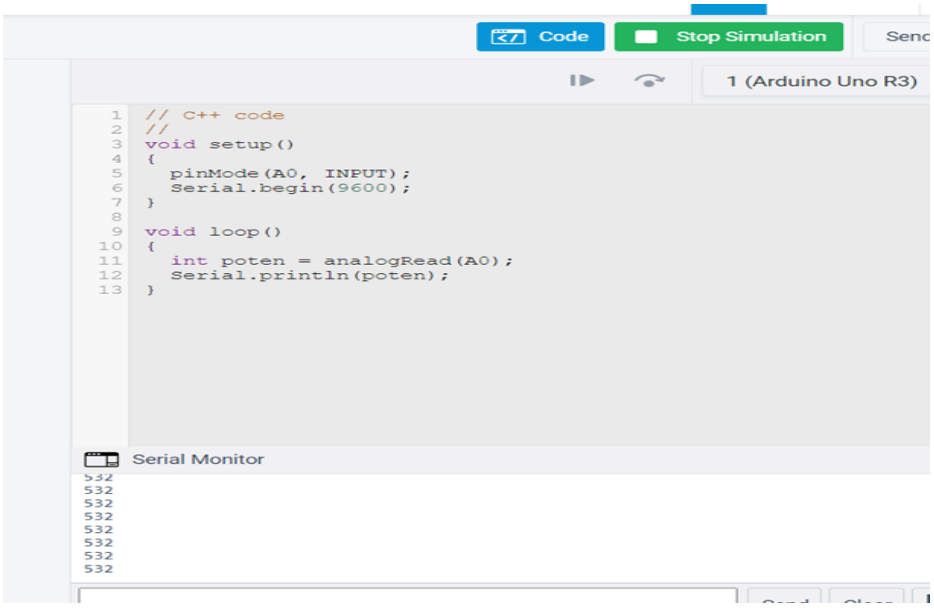


<u>บทต่อไป</u>เราจะเราจะมาลองใช้การต่อวงจร Arduino simulation ของ tinkercad เราไป simulate circuit สิ่งที่เตรียมใน simulation

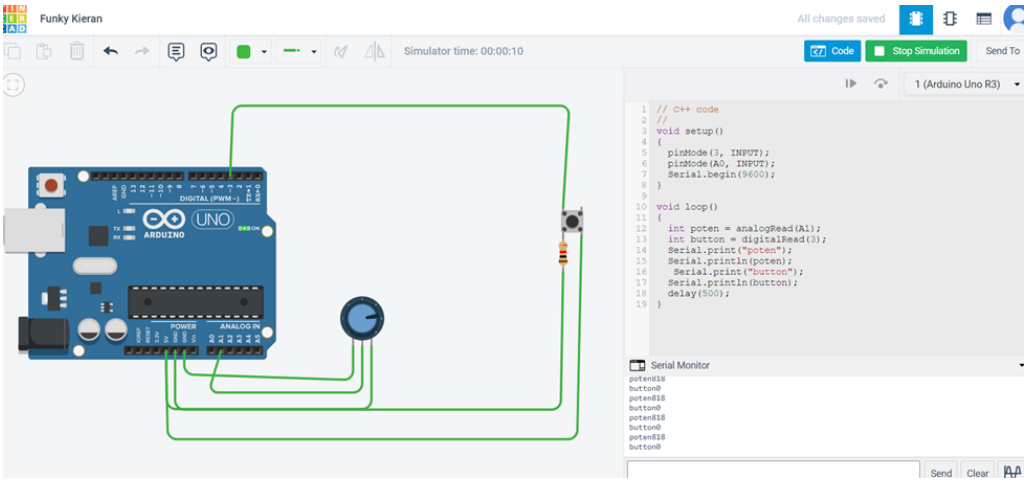
- Arduino
- Multimeter
- Potentiometer
- Button



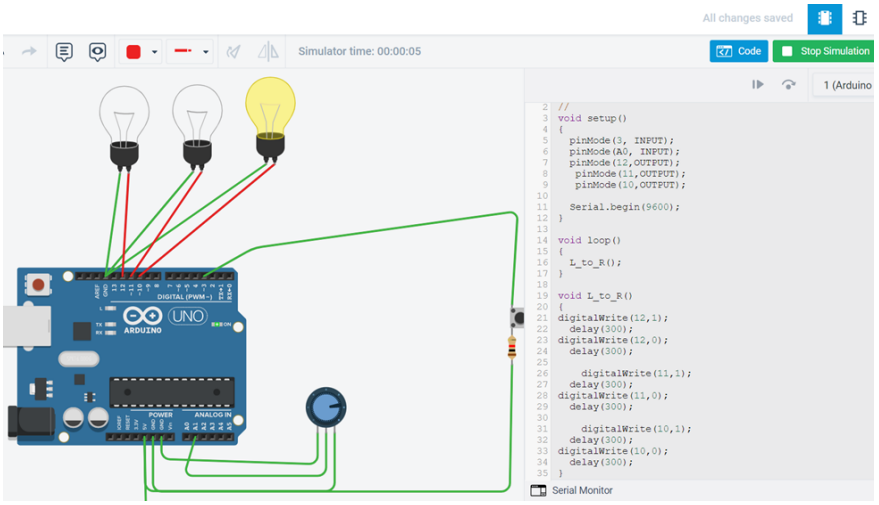
เราจะลองการใช้อ่านค่า potential จากการลองการเขียนโค้ดในตัวของ Arduino ดูว่าจะได้ค่ามาได้เท่าไหร่ ตามการหมุนของ potentiometer ค่าที่อ่านได้ออกจะได้มา 0-1023 โดยค่าที่ออกมาได้จะเป็นค่าของ Analog



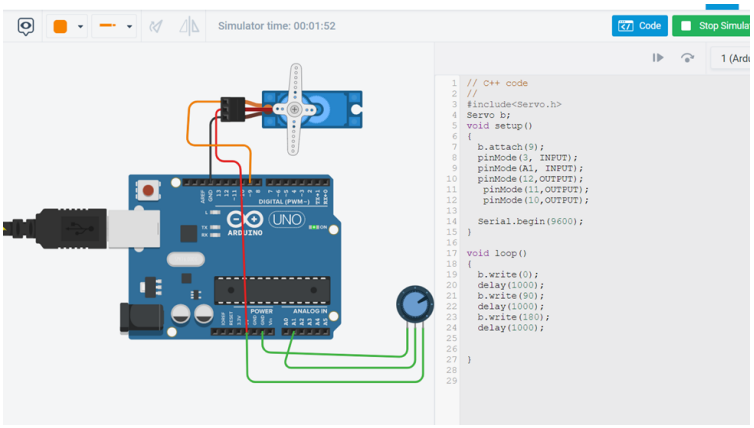
เราจะลองอ่านค่า digital ของ port 3 โดยใช้ ตัวของ button เข้ามาช่วยทำการดูค่า digital เป็น 0กับ1เมื่อกด button นั้น



ต่อไปเราลองต่อ output ออกมาทำวงกรโดยลองใช้ หลอดไฟดู ทำวงจรของไฟวิ่งจากซ้ายไปขวาทำการเปิดปิดและใส่ delay จากโค้ดตัวอย่าง



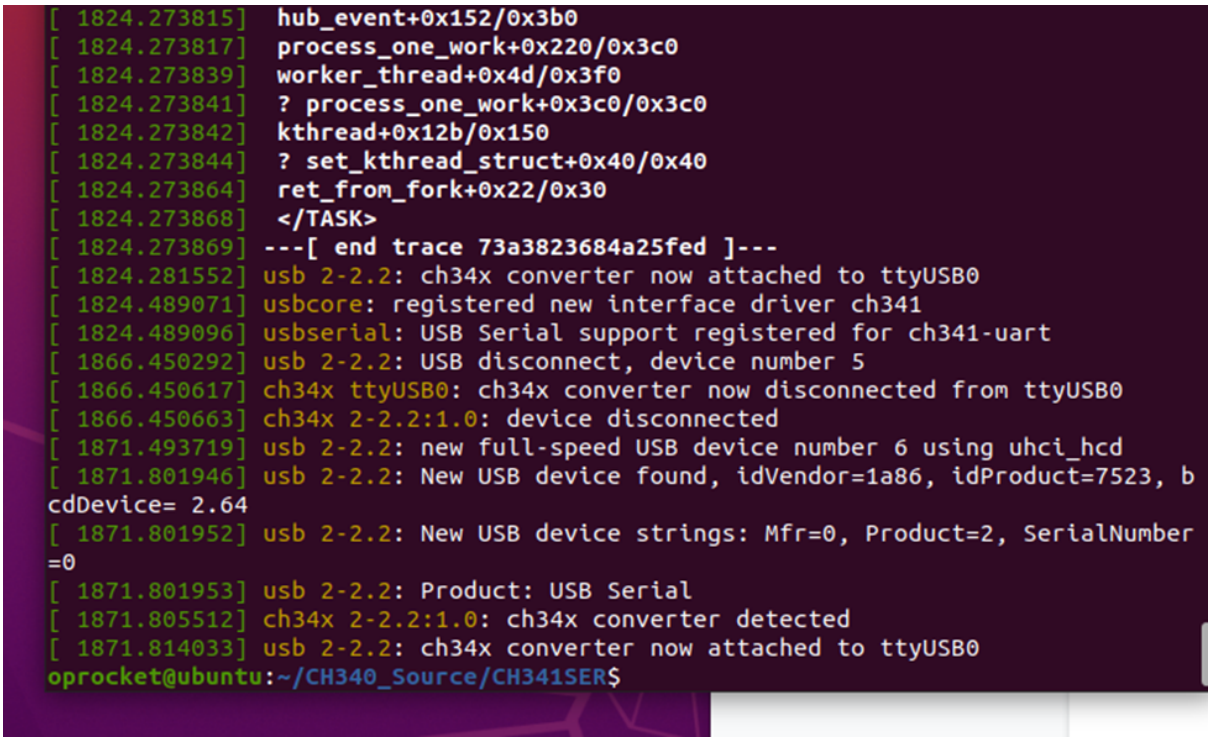
ตัวอย่างการต่อ servoของ Arduino โดยเราจะเรียกใช้ library ของ servo และเราจะลองทำการกำหนดมุมของ servo ต่างๆจากโค้ดรูปภาพด้านล่าง



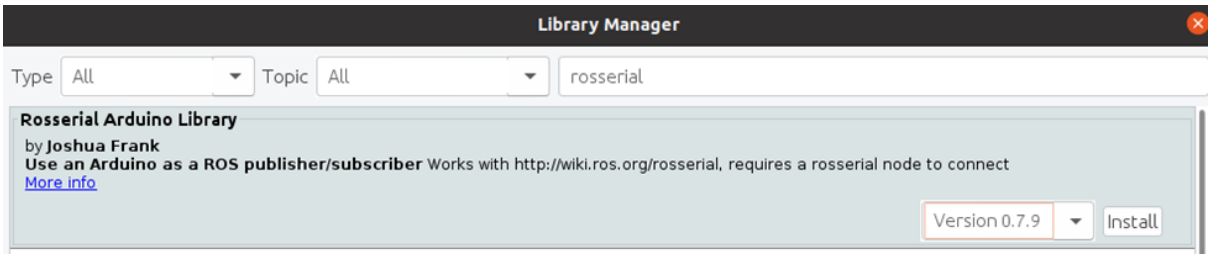
<u>บทต่อไป</u>เราจะไปลง Arduino บน platform ของ ubuntu เพื่อทำการควบคุม bord arduinoของเรา โดยขั้นตอน install Arduino ขั้นแรกทำการใส่ คำสั่งตามขั้นตอนต่อไปนี้นี้ลงไปที่ terminal ตามขั้นตอน

```
sudo apt-get update
sudo apt-get upgrade
mkdir arduino
cd arduino
wget http://downloads.arduino.cc/arduino-1.8.15-linux64.tar.xz
tar -xvf ./arduino-1.8.15-linux64.tar.xz
cd arduino-1.8.15
sudo ./install.sh
```

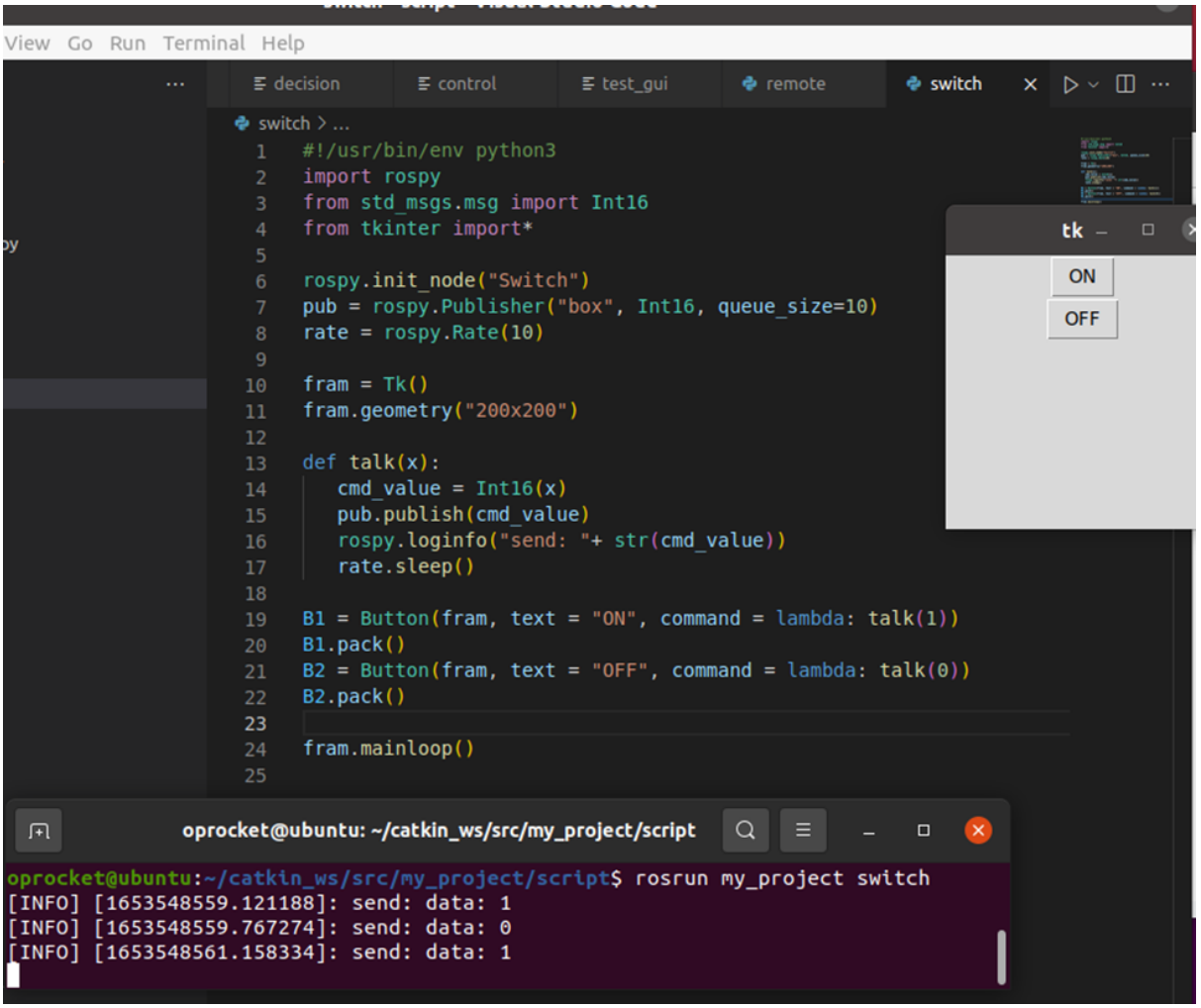
จากนั้นเมื่อเราลองต่อ bord Arduino เข้ากับ pc ของเราแล้วเราจะทำการ เช็ตการเชื่อด้วยคำสั่ง dmesg ลงใน terminal ถ้าเชื่อมต่อสำเร็จจะขึ้นตามหน้า terminal ตามรูปด้านล่าง



หลังจากที่เราได้โปรแกรม Arduino มาแล้ว เราก็เลือกport และbord ให้ตรงกับรุ่นที่เราใช้ จากนั้นให้เพิ่ม library ของ ROS ให้ install ตามรูปด้านล่างนี้



เมื่อเราได้ libraryแล้วเราจะลองเขียนโปรแกรม เอาไว้ควบคุม จากการสร้าง GUI ครั้งก่อน จากตัวของ virtual studio โดยตัวอย่างโค้ดที่เราจะใช้ ขั้นแรกเราจะลองทำตัวของปุ่มเป็น publisher ส่งค่าออกไปได้จากการกดดังรูปด้านล่าง

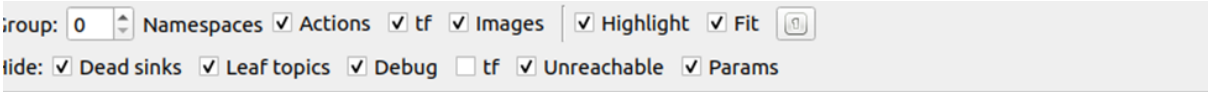


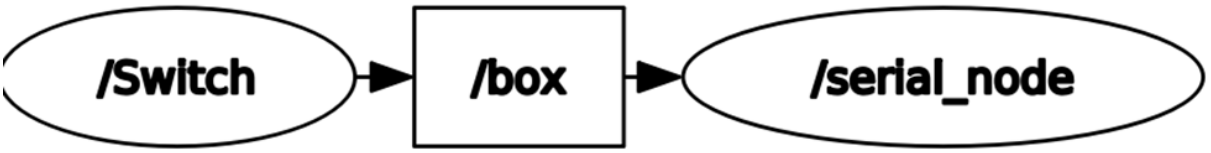
ต่อไปเราจะสร้างอีก1node เป็น node ของตัว Arduino โดย node นี้ต้องเชื่อมกับ node switch ของเราที่เราสร้างขึ้นมาพร้อมกับตัว GUI โดยการสร้าง node ในตัวของ Arduino และเป็นตัว subscribe ตัวของที่เราสร้างไว้ตามตัวเอง publisher ของเราชื่อ box เราก็จะ subscribe มาจากโค้ดตามตัวอย่างด้านล่าง

onoffnode §

```
#include <ros.h>
#include<std_msgs/Int16.h>


void decision(const std_msgs::Int16 & cmd_msg)
{

  int cmd =cmd_msg.data;

  digitalWrite(13,cmd);
}

ros::NodeHandle nh;
ros::Subscriber<std_msgs::Int16> sub("box", decision);
void setup()
{
  pinMode(13,OUTPUT);
  nh.initNode(); nh.subscribe(sub);
}
void loop()
{
  nh.spinOnce(); delay(1);
}
```

เมื่อเรารันตัวของ switch และตัวของ subscriber ใน arduino แล้วการอ่านค่า rqt_graph ต้องเป็นแบบนี้เป็นตัวอย่างประมาณนี้เป็นอันเสร็จสิ้น





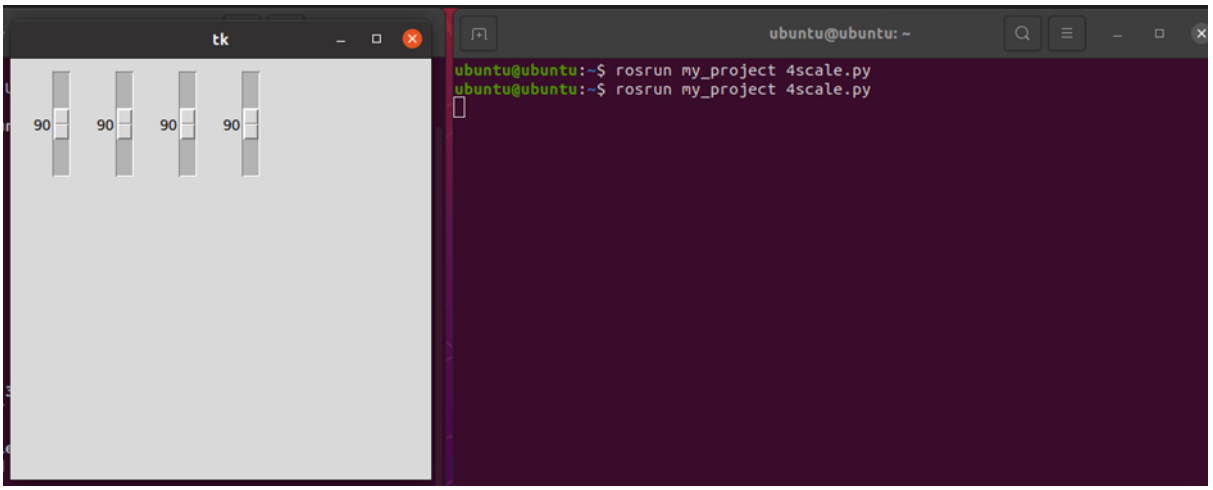
บทที่5

ใช้ทั้งหมดที่เรียนกับ ROBOT ARM

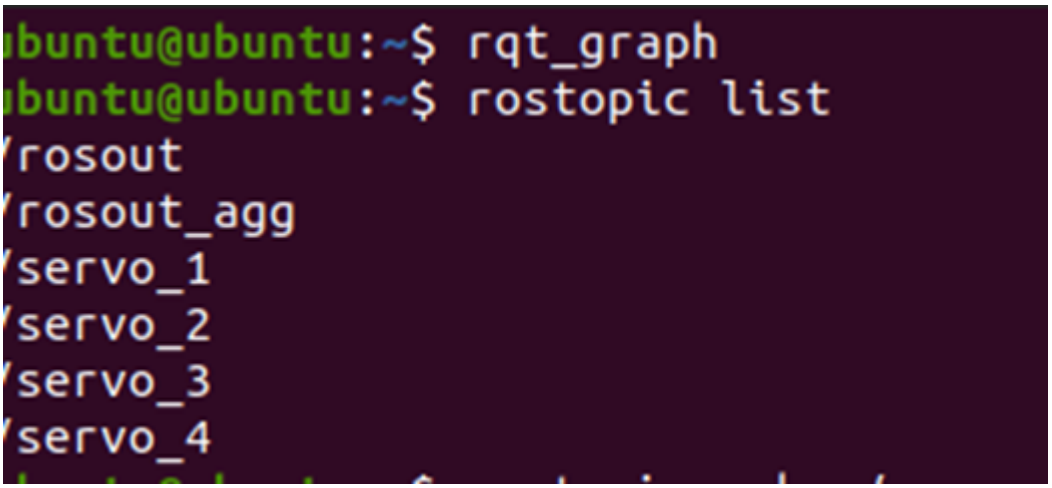
ทำการสร้างไฟล์ขึ้นมาชื่อ 4scale.py ในโฟลเดอร์ script และวางโค้ดลงไป

```python
#!/usr/bin/env python3
from tkinter import*
import rospy
from std_msgs.msg import Int16

pub1 = rospy.Publisher('servo_1', Int16, queue_size=10)
pub2 = rospy.Publisher('servo_2', Int16, queue_size=10)
pub3 = rospy.Publisher('servo_3', Int16, queue_size=10)
pub4 = rospy.Publisher('servo_4', Int16, queue_size=10)
rospy.init_node('GUI', anonymous=True)
rate = rospy.Rate(100)

frame=Tk()
frame.geometry("400x400")

S1_value = 0
S2_value = 0
S3_value = 0
S4_value = 0

def talker(val):
    global S1_value
    global S2_value
    global S3_value
    global S4_value

    S1_value = Int16(S1.get())
    S2_value = Int16(S2.get())
    S3_value = Int16(S3.get())
    S4_value = Int16(S4.get())

    pub1.publish(S1_value)
    pub2.publish(S2_value)
    pub3.publish(S3_value)
    pub4.publish(S4_value)
    rate.sleep()

S1= Scale(frame,from_=0,to=180,command=talker)
S1.place(x=10, y=10)
S1.set(90)

S2= Scale(frame,from_=0,to=180,command=talker)
S2.place(x=70, y=10)
S2.set(90)

S3= Scale(frame,from_=0,to=180,command=talker)
S3.place(x=130, y=10)
S3.set(90)

S4 = Scale(frame,from_=0,to=180,command=talker)
S4.place(x=190, y=10)
S4.set(90)
frame.mainloop()
```

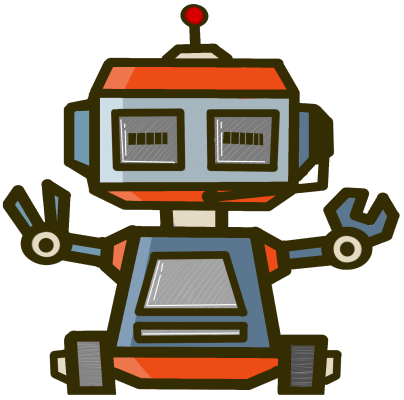
เมื่อกด run จะได้หน้าต่างแบบนี้ ***ต้อง run roscore ในอีก terminalก่อน***

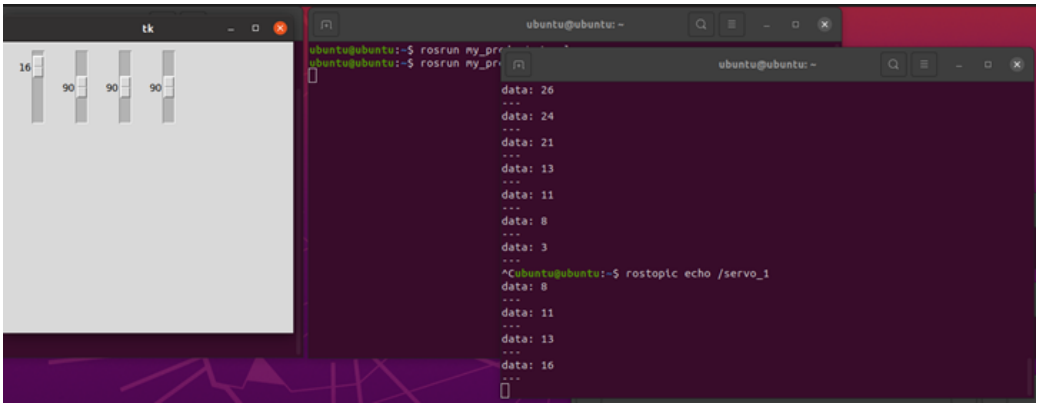


ใส่คำสั่ง rostopic list เพื่อจะดูว่ามี topic อะไรบ้าง



จากนั้นเราจะดูค่าของ topic servo_1 โดยใช้คำสั่ง rostopic echo servo_1 เมื่อเลื่อน scale จะพบว่าค่าจะขึ้นตามscaleที่เลื่อน

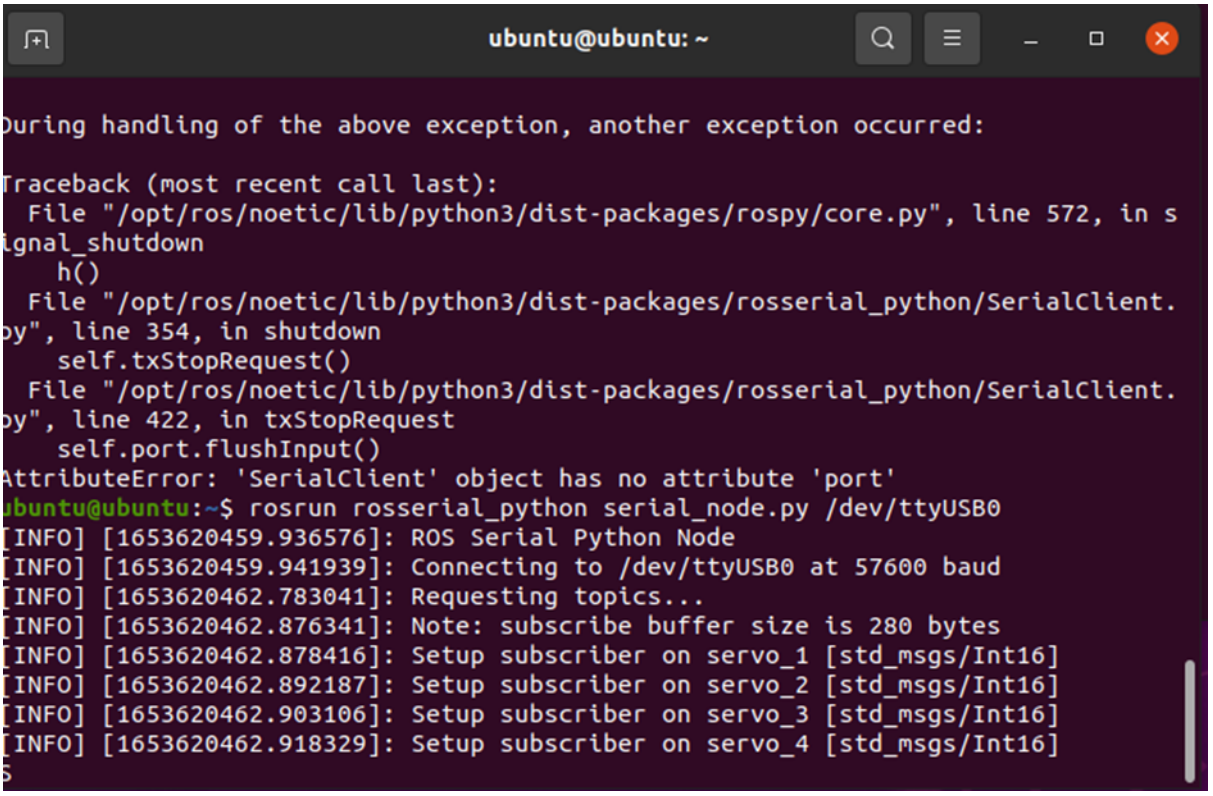




ทำการเปิดโปรแกรม Arduino IDE ขึ้นมาและสร้างไฟล์ ชื่อ 4motor ในโฟลเดอร์และวางโค้ดนี้ลงไป

```
#include <Servo.h>
#include <ros.h>
#include <std_msgs/Int16.h>


Servo servo_1;
Servo servo_2;
Servo servo_3;
Servo servo_4;

void servo_cb_1( const std_msgs::Int16& cmd_msg){
  servo_1.write(cmd_msg.data); //set servo angle, should be from 0-180
}

void servo_cb_2( const std_msgs::Int16& cmd_msg){
  servo_2.write(cmd_msg.data); //set servo angle, should be from 0-180
}

void servo_cb_3( const std_msgs::Int16& cmd_msg){
  servo_3.write(cmd_msg.data); //set servo angle, should be from 0-180
}

void servo_cb_4( const std_msgs::Int16& cmd_msg){
  servo_4.write(cmd_msg.data); //set servo angle, should be from 0-180
}
ros::NodeHandle  nh;

ros::Subscriber<std_msgs::Int16> sub_1("servo_1", servo_cb_1);
ros::Subscriber<std_msgs::Int16> sub_2("servo_2", servo_cb_2);
ros::Subscriber<std_msgs::Int16> sub_3("servo_3", servo_cb_3);
ros::Subscriber<std_msgs::Int16> sub_4("servo_4", servo_cb_4);


void setup(){

  nh.initNode();
  nh.subscribe(sub_1);
  nh.subscribe(sub_2);
  nh.subscribe(sub_3);
  nh.subscribe(sub_4);

  servo_1.attach(9);
  servo_2.attach(10);
  servo_3.attach(11);
  servo_4.attach(12);
}

void loop(){
  nh.spinOnce();
  delay(1);
}

```

จากนั้นเราจะรันตัวส่ง ใน terminal เพื่อส่งค่าเชื่อมกับ บอร์ดนั้นไปกับตัวของ Arduino



เราได้ทำการทดสอบโปรแกรมในกาควบคุมข้อต่อของแขนหุ่นเป็นอันเสร็จสิ้น.