

คู่มือวิชา Robot Operating System (ROS)

จัดทำโดย

นาย ชินวัตร ซาเล

นาย ปณชัย ทองไพรวรรณ

นาย ชนาเทพ ศีลาอาสน์

จาก

คณะ วิศวกรรมศาสตร์ ภาควิชา วิศวกรรมการผลิตและหุ่นยนต์

สาขาวิชา วิศวกรรมหุ่นยนต์และระบบอัตโนมัติ

มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ(KMUTNB)

นำเสนอ

สถาบันพัฒนาบุคลากรสาขาเทคโนโลยีการผลิตอัตโนมัติและหุ่นยนต์



คู่มือนี้จัดทำขึ้นเพื่อช่วยให้ผู้อ่านได้เข้าใจ ROS มากขึ้น

สิ่งที่ผู้สอนและผู้เรียนต้องเตรียม

- คอมพิวเตอร์/โน้ตบุค
- อินเทอร์เน็ต
- Ubuntu

แผนการสอน

บทที่1 ลงโปรแกรม ROS และอธิบายพื้นฐาน ROS

บทที่2 ลงโปรแกรม Python และอธิบายพื้นฐาน Python

บทที่3 เรียนการทำ database cloud และการสร้าง GUI

บทที่4 ใช้ arduino สื่อสารกับระบบ ROS และก็มีการสร้าง simulation ใน tinker cad

บทที่5 ได้ใช้ทั้งหมดที่เรียนกับ ROBOT ARM



บทที่ 1

ROS

เราจะมาทำความเข้าใจก่อนว่า Robot Operating System หรือว่า ROS นั้นคืออะไรเอาไปใช้ได้ยังไงแล้วประโยชน์คืออะไรสามารถนำไปใช้ในชีวิตประจำวันได้หรือไม่

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



ROS ได้รับการพัฒนาจากหลากหลายหน่วยงาน สำหรับการพัฒนาหุ่นยนต์หลากหลายรูปแบบ แม้ว่าข้อมูลทั้งหมดจะอยู่บนเซิร์ฟเวอร์เดียวกันเพื่อให้ผู้ที่สนใจเข้าถึงได้ง่ายในช่วงหลายปีที่ผ่านมา แต่ไม่นานรูปแบบของการเก็บข้อมูลบนเซิร์ฟเวอร์แยก ได้ถูกนำมาใช้เป็นจุดแข็งสำหรับกลุ่มคนที่ใช้ ROS โดยพวกเขาสามารถที่จะเป็นเก็บ source code ของตัวเองบนเซิร์ฟเวอร์ของตัวเองได้ และยังสามารถรักษาความเป็นเจ้าของและการควบคุมการเข้าถึง source code ได้เต็มรูปแบบ โดยไม่ต้องขออนุญาตจากใคร ถ้าหากพวกเขาต้องการเผยแพร่ข้อมูลเหล่านั้นต่อสาธารณะพวกเขาสามารถทำได้และเป็นที่ยอมรับในหมู่คนที่ใช้ ROS ด้วยกัน พวกเขาอาจได้รับคำตอบแทนที่พวกเขาสมควรได้รับสำหรับความสำเร็จของพวกเขา และได้ประโยชน์จากข้อเสนอแนะทาง

เทคนิคเพื่อการปรับปรุงซอฟต์แวร์ของพวกเขาให้ดีขึ้น เช่นเดียวกับโครงการซอฟต์แวร์โอเพนซอร์สต่างๆ

ในขณะนี้ชุมชนของผู้ใช้ ROS ประกอบด้วยผู้ใช้นับหมื่นทั่วโลก ตั้งแต่โครงการงานอดิเรกเล็กๆ ไปจนถึงระบบอัตโนมัติอุตสาหกรรมขนาดใหญ่ ส่วนแล้วแต่ใช้ ROS ในการสร้างระบบของหุ่นยนต์

ROS Concept

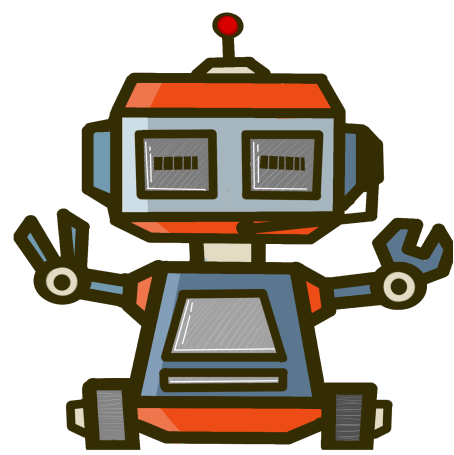
ระบบ ROS จะแบ่ง Concept ของระบบออกเป็น 3 ส่วน คือ

1. ROS Computation Graph Level

คอนเซ็ปของระบบ ROS ก็คือการแบ่งโมดูลการทำงานของหุ่นยนต์ออกเป็นส่วนๆ แล้วให้แต่ละส่วนส่งข้อความคุยกันผ่านทาง Topic ต่างๆในรูปแบบของ Peer-to-Peer กับส่วนอื่นๆ โดยจะมี ROS Master เป็นตัวบริหารจัดการทำงานของระบบทั้งหมด ดังนั้นจะมีคำศัพท์ที่ควรจะต้องรู้ไว้ก็คือ 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 ความรู้ โหลด packages ต่างๆมาใช้งาน อีกทั้งยังใช้เป็นแหล่งเผยแพร่ packages ที่เราออกแบบ

ซึ่งคู่มือนี้เราจะเป็นการเน้นเนื้อหาBasicต่างๆของ ROS ให้ผู้อ่านมีความรู้ความเข้าใจใน พื้นฐานของ ROS ดังนั้นเราเน้นไปทางด้านของ ROS Computation Graph Level โดยจาก บทความconceptด้านบน เราจะมีอธิบายเพิ่มเติมต่อคำศัพท์ต่างๆของROSให้มีความเข้าใจได้ มากขึ้น

- **Node** :เป็นกระบวนการที่ใช้ในการดำเนินการ โปรแกรมต่าง 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 โดยอาจจะอยู่ในรูปแบบของข้อความ ค่ามาตรฐาน (จำนวนเต็ม, จุดลอย, บูลีน, ฯลฯ) ข้อมูลอาร์เรย์ต่างๆไป รวมไปถึงข้อมูลอาร์เรย์ที่มีความซับซ้อน

- **Topics** : ข้อความจะถูกส่งผ่านระบบการขนส่งที่มีการ publish / subscribe โหนดที่ส่งข้อความจะเรียกว่า publishing ชื่อที่ระบุใน Topic จะใช้เพื่อระบุเนื้อหาของข้อความ โหนดที่สนใจในข้อมูลบางประเภทจะทำการ subscribe หัวข้อที่ต้องการ อาจมีการ publish / subscribe หลายรายพร้อมกันสำหรับหัวข้อเดียวและ โหนดเดียวอาจ 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 กัน โดยขั้นแรก เราจะมาติดตั้งตัวของ ROS ก่อน เนื่องจากระบบนี้จำเป็นต้องติดตั้ง ระบบ Linux กันสักก่อน มีให้เลือกสองรูปแบบ **Ubuntu & 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 โหลดสคริปไฟล์อัปเดตเสร็จ ก็ทำการ Update Linux ดังนี้

```
sudo apt update
```

ขั้นตอนถัดไป มีการติดตั้งโปรแกรมหรือฟังก์ชันเสริมต่างๆ ให้เลือกสำหรับผู้เริ่มต้นผมขอแนะนำลงไปทั้งหมดเลยครับ

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

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

ขั้นตอนถัดไป มีการติดตั้งโปรแกรมของPython เชื่อก 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 roserial_arduino
```

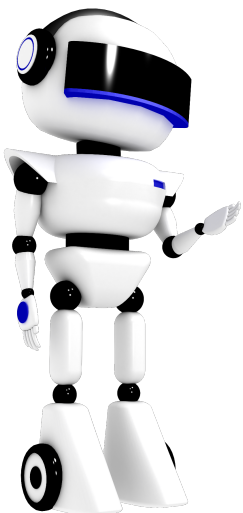
```
cd ..
```

```
catkin_make
```

```
sudo snap install code --classic
```

if you want to know more information:

<http://wiki.ros.org/noetic/Installation/Ubuntu>



บทที่ 2

Python

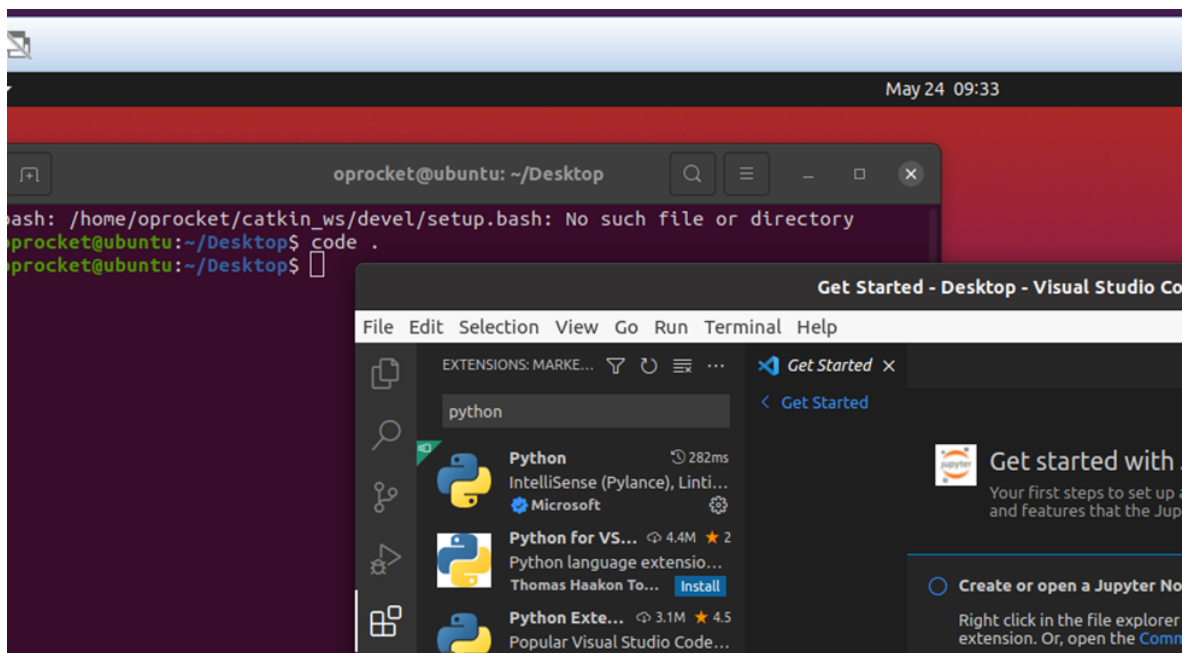
เริ่มแรกเราจะสร้าง python ใน ubuntu โดยขั้นแรกเราจะทำการโหลดโปรแกรม Visual Studio Code ผ่านทาง terminal (Ctrl + Alt + T) โดยคำสั่งพิมพ์ใน terminal

: sudo snap install code – classic

หลังจากโหลด visual studio เสร็จแล้ว จะเปิดโปรแกรมจาก terminal ด้วยคำสั่ง **code .**

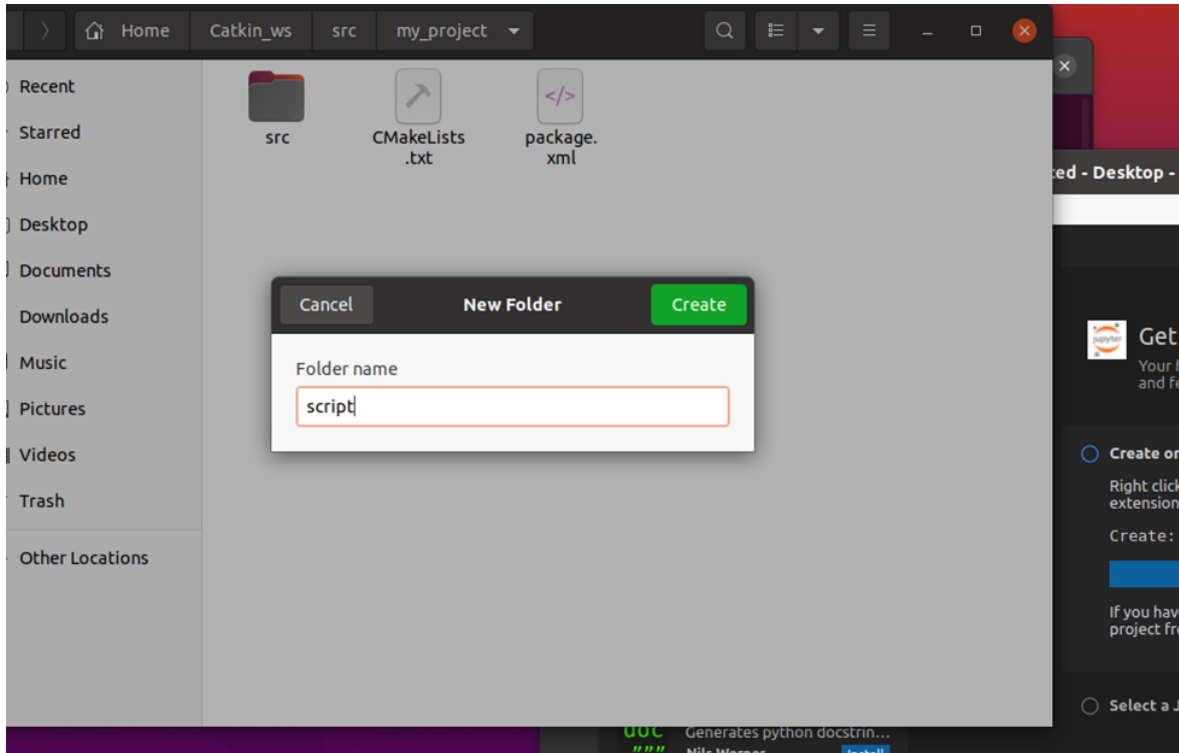
หลังจากเปิดโปรแกรมมาแล้วให้ติดตั้ง python ในโปรแกรม visual studio โดยไปที่ environment แล้ว search ว่า python

หลังจากนั้น install python มาดังภาพด้านล่าง



คำสั่งเปิดใช้งาน vscode = code .

หลังจากนั้นเราจะสร้าง folder script เพื่อที่จะใส่ไฟล์ python เราสร้างขึ้นมา script ตามรูปภาพด้านล่าง



ภาพแสดงการสร้าง โฟลเดอร์

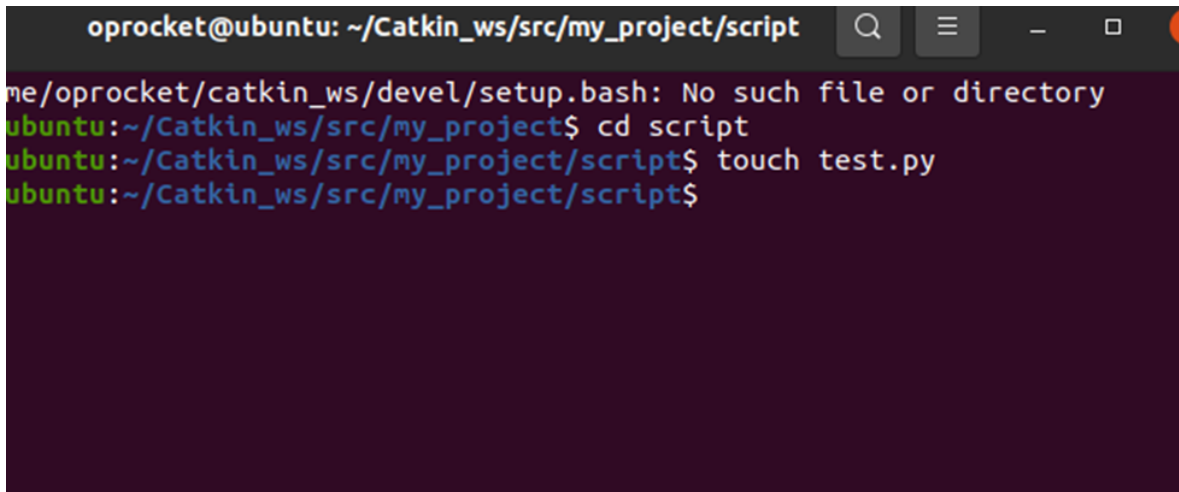
หลังจากนั้นเราจะสร้างไฟล์ .py โดยสร้างจาก terminal ด้วยคำสั่ง touch ตามด้วยชื่อไฟล์.py ตัวอย่างเช่น touch test.py ตามรูปด้านล่าง

```
oprocket@ubuntu: ~/Catkin_ws/src/my_project/script
~/oprocket/catkin_ws/devel/setup.bash: No such file or directory
ubuntu:~/Catkin_ws/src/my_project$ cd script
ubuntu:~/Catkin_ws/src/my_project/script$ touch test.py
ubuntu:~/Catkin_ws/src/my_project/script$
```

คำสั่งสร้างไฟล์

เมื่อมีไฟล์.py เรียบร้อยแล้วหลังจากนั้นเราจะเขียนลงpython ลงไปไฟล์นั้นผ่านโปรแกรม visual studio โดยเปิดไฟล์เราสร้างขึ้นมาจากนั้นหัวข้อให้พิมพ์ว่า

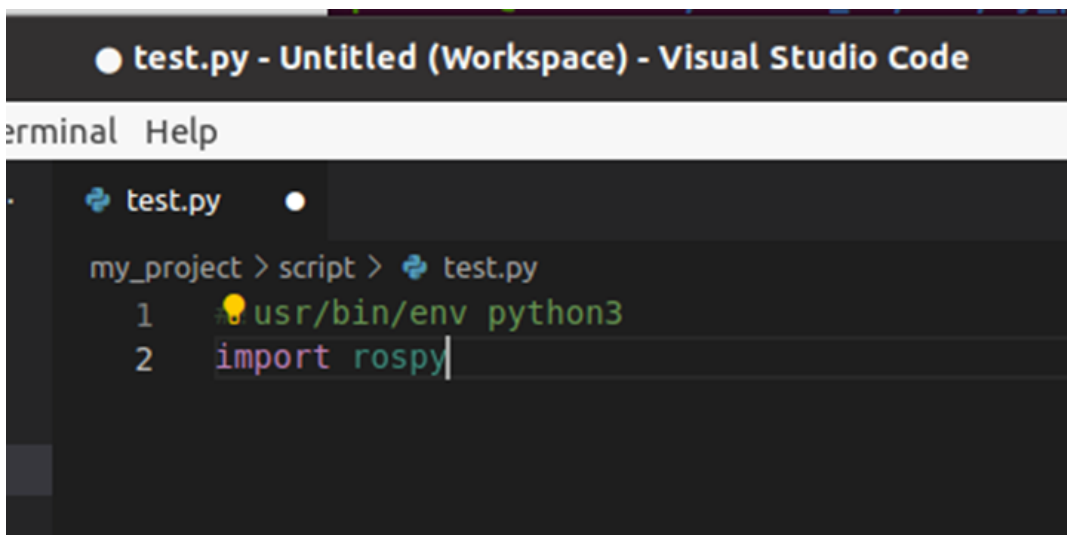
```
#!/usr/bin/env python3
```



```
oprocket@ubuntu: ~/Catkin_ws/src/my_project/script
me/oprocket/catkin_ws/devel/setup.bash: No such file or directory
ubuntu:~/Catkin_ws/src/my_project$ cd script
ubuntu:~/Catkin_ws/src/my_project/script$ touch test.py
ubuntu:~/Catkin_ws/src/my_project/script$
```

ภาพการสร้างไฟล์ python

เมื่อไหร่ก็ตามที่เราต้องการจะใช้เพื่อใช้ร่วมงานกับ ROS เราจะใส่หัวข้อ `import rospy` ขึ้นเสมอ ไม่งั้นจะไม่มีคำสั่งในการเขียนโปรแกรมผ่านทาง ROS



```
test.py - Untitled (Workspace) - Visual Studio Code
Terminal Help
test.py
my_project > script > test.py
1  #!usr/bin/env python3
2  import rospy
```

คำสั่งที่ต้องมีตลอดเมื่อใช้ ROS คู่กับ python

เราจะทำการทดลองเขียน python เป็นคำสั่ง basic ดูว่าเราจะรันได้ไหม

```
test.py - oprocket - Visual Studio Code
minal Help
test.py x
Catkin_ws > src > my_project > script > test.py > ...
1  #!/usr/bin/env python3
2  import rospy
3  print("Hello world")
4  user_name = input("username: ")
5  pass_word = input("Password: ")
6  if(user_name == 'admin'):
7      print('user correct')
8      if(pass_word == '1234'):
9          print('password correct')
10         else:
11             print('password incorrect')
12     else:
13         print('user incorrect')
14
15
```

ภาพทดสอบว่าโปรแกรมทำงานได้ไหม

โดยการรันเราจะทำการรันไฟล์ python ที่เราเขียนผ่านทางตัวของ terminal ของubuntu

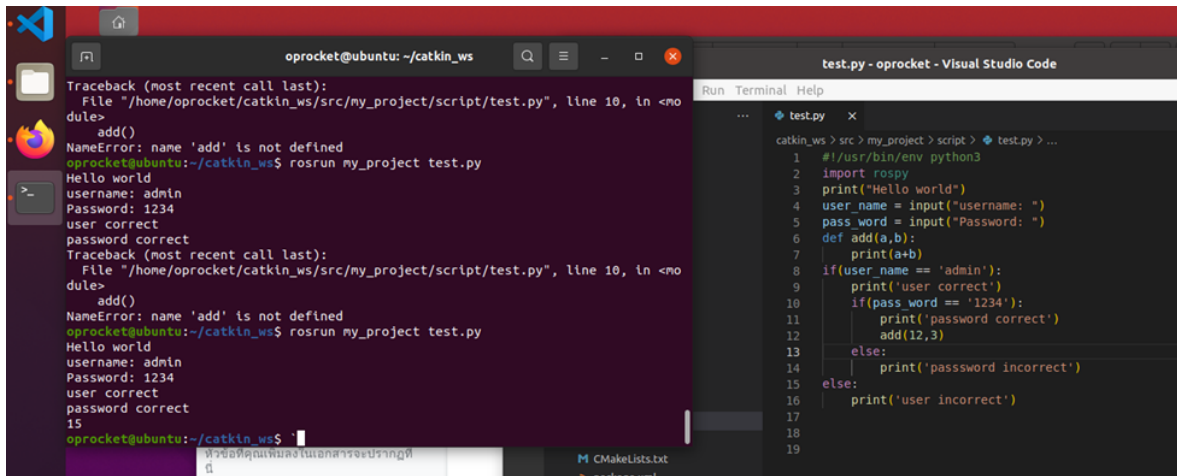
ด้วยการรันผ่าน package ที่เราสร้างขึ้นมา โดยคำสั่ง rosrn ตามด้วยชื่อแพคเกจของเราแล้วตามด้วยชื่อไฟล์ py

ตัวอย่าง rosrn my_project test.py

```
rc/my_project
[rosrn] Found the following, but they're either not files,
[rosrn] or not executable:
[rosrn] /home/oprocket/catkin_ws/src/my_project/script/test.py
oprocket@ubuntu:~/catkin_ws$ rosrn my_project test.py
Hello world
username: admin
Password: 1234
user correct
password correct
oprocket@ubuntu:~/catkin_ws$
```

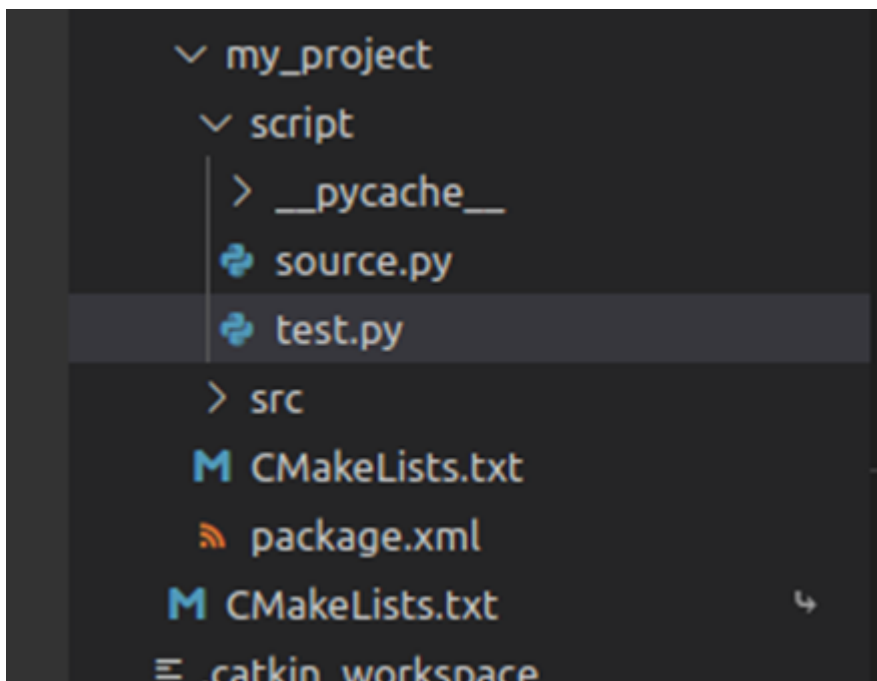
ตัวอย่าง rosrn my_project test.py

Code python ตัวอย่างสำหรับ run ในterminal แต่เขียนในcodeในvisual studio

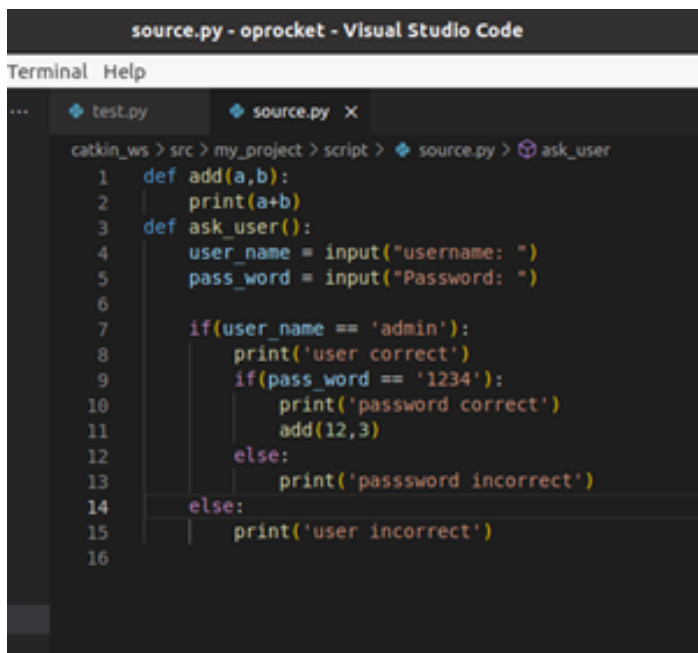


Code ตัวอย่าง
การสร้างไฟล์ที่ชื่อ source.py

ต่อไปเราจะลองสร้างการทำไฟล์แยกแต่สร้างความสัมพันธ์กันได้ โดยขั้นแรก
เราจะสร้างไฟล์ .py ใหม่ ตัวอย่างเราจะสร้างไฟล์ชื่อ source.py



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



```
source.py - oprocket - Visual Studio Code
Terminal Help
test.py source.py X
catkin_ws > src > my_project > script > source.py > ask_user
1 def add(a,b):
2     print(a+b)
3 def ask_user():
4     user_name = input("username: ")
5     pass_word = input("Password: ")
6
7     if(user_name == 'admin'):
8         print('user correct')
9         if(pass_word == '1234'):
10            print('password correct')
11            add(12,3)
12        else:
13            print('password incorrect')
14    else:
15        print('user incorrect')
16
```

การสร้างฟังก์ชันที่ชื่อว่า `add`

ในตัว `test.py` จะเรียกค่า `def` จาก `source.py` โดยพิมพ์คำสั่ง `from ...` แล้วพิมพ์ตัวแปรของ `def` ที่เราสร้าง เป็นการเสร็จสิ้น

```
test.py - oprocket - Visual Studio Code
nal Help
test.py x source.py
catkin_ws > src > my_project > script > test.py
1  #!/usr/bin/env python3
2  import rospy
3  from source import*
4  print("Hello world")
5  ask_user()
```

คำสั่งเรียกใช้ฟังก์ชันจากไฟล์ที่ชื่อว่า source.py

บทต่อไป เราจะทำการยกตัวอย่าง while/for loop ของ python

สำหรับ while loop ทำการสร้าง loop ในตัวอย่างเราจะทำ loop ที่ทั้งหมด 10 ครั้ง โดยการตั้ง while(x<=10) แล้ว x ที่เราสร้างนั้นเริ่มต้นจาก 1

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

ตัวอย่างโค้ด

```
15
x-value: 0
finish while
x-value: 1
finish while
x-value: 2
finish while
x-value: 3
finish while
x-value: 4
finish while
x-value: 5
finish while
x-value: 6
finish while
x-value: 7
finish while
x-value: 8
finish while
x-value: 9
finish while
x-value: 10
finish while
```

ผลลัพธ์ของโปรแกรม

ต่อสำหรับ For loop จะคล้ายๆ กับ while loop แต่จะสั้นกว่าทำให้การเขียนนั้นง่ายมากยิ่งขึ้น โดยข้อเสียคือการสร้าง condition ในการนับนั้นจะไม่สามารถทำได้เท่ากับ while loop

ในตัวอย่างเราจะลองใช้ for loop โดยการนับกลับหลังจาก range 5-0 กลับหลังโดยพิมพ์ -1 ไปใน condition

```

        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 ชั้นแรกเราจะทำการสร้างnodeขึ้นมาด้วยคำสั่งของ rospy ตามตัวอย่างรูปด้านล่าง เราจะให้ชื่อว่า node_one.

```
catkin_ws > src > my_project > script > test.py
2 import rospy
3 from source import*
4
5 rospy.init_node("node_one")
```

การสร้าง node ของ ros

สำหรับการใช้ROSเราควรกำหนด rateของrosเพื่อไม่ให้ CPUของคอมพิวเตอร์
ไม่ให้ทำงานหนักเกินไป

และเราทำwhile loopเพื่อที่จะเช็คว่เครื่องนั้นถูกปิดอยู่หรือป่าว

```
5 rospy.init_node("node_one")
6 rospy.sleep(0.1)
7 rate = rospy.Rate(10)
8 while(not rospy.is_shutdown()):
9     rate.sleep
```

ตัวอย่างโค้ดเพื่อเช็คว่เครื่องเปิดอยู่หรือเปล่า

หลังจากที่เราสร้างnodeแล้วเราจะลองสั่ง roscore ดูว่ ที่เราทำนั้นสามารถเป็น
ตัว subscribe ได้หรือป่าว

เมื่อรันทั้ง roscore และตัว pythonที่เราสร้างแล้ว จะเช็คตัวnode ด้วยคำสั่ง

roscall list ในตัวของ terminal

ตัวอย่างรูปด้านล่างจะเจอทั้ง 2 nodeถือเป็นอันถูกต้อง

```
oprocket@ubuntu:~$ rosnodet list
/node_one
/rosout
oprocket@ubuntu:~$
```

ภาพรายชื่อ node

ต่อไปเป็นการทำเช็คดูว่าตัวอักษรของ ROS นั้นเป็นสีอะไรจาก ตัวของ info/warning/error จากการพิมพ์

คำสั่ง ของ rospy.log/warn/err แล้วแต่จะคำสั่งจะมีสีของตัวอักษรนั้นๆตามรูปด้านล่าง

```
rospy.log/warn/err
rospy.init_node("node_one")
File "/opt/ros/noetic/lib/python3/dist-packages/rospy/client.py", line 336, in
init_node
    raise rospy.exceptions.ROSInitException("Failed to initialize time. Please c
heck logs for additional details")
rospy.exceptions.ROSInitException: Failed to initialize time. Please check logs
for additional details
oprocket@ubuntu:~/catkin_ws$
oprocket@ubuntu:~/catkin_ws$ rosrund my_project test.py
Unable to register with master node [http://localhost:11311]: master may not be
running yet. Will keep trying.
shutdown request: [/node_one] Reason: new node registered with same name
oprocket@ubuntu:~/catkin_ws$ rosrund my_project test.py
^Coprocket@ubuntu:~/catkin_ws$
oprocket@ubuntu:~/catkin_ws$ rosrund my_project test.py
[INFO] [1653375162.121114]: start node_one
[WARN] [1653375162.122773]: this is warning
[ERROR] [1653375162.123956]: this is error RED

import rospy
from numpy import rate
from source import*
if __name__ == "__main__":
    rospy.init_node("node_one")
    rospy.sleep(0.1)
    rate = rospy.Rate(10)
    rospy.loginfo("start node_one")
    rospy.logwarn("this is warning")
    rospy.logerr("this is error RED")
    while(not rospy.is_shutdown()):
        rate.sleep
```

สีจะขึ้นอยู่กับคำสั่งที่เราใส่ไป

ต่อไปเราจะลองกับตัว turtlesim ที่คำสั่งในpython ขึ้นแรกเราจะเขาไปเช็ค ว่าไฟล์ .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$
```

ภาพเข็คิดว่าไฟล์สามารถรันได้ใหม่

หลังจากนั้นเราจะเขียนโค้ดpythonเพื่อควบคุมเต่าผ่านตัวของ ROS เริ่มจากการเขียนโค้ดpython ในไฟล์.pyที่เราสร้างขึ้นมา
เราก็จะสร้างnodeขึ้นมาปกติ

```
package.xml  draw_circle.py
script > draw_circle.py
1  #!/usr/bin/env python3
2  import rospy
3  from geometry_msgs import Twist
4
5  if __name__ == "__main__":
6      rospy.init_node("draw circle")
7      rospy.loginfo("node draw_circle is started")
8
```

โค้ดตัวอย่างการสร้างnode

หลังจากนั้นเราทำให้nodeนี้เป็นpublisher ไว้คอยส่งค่าออกไปที่ subscriber
จากการเขียนโค้ดต่อไปนี้

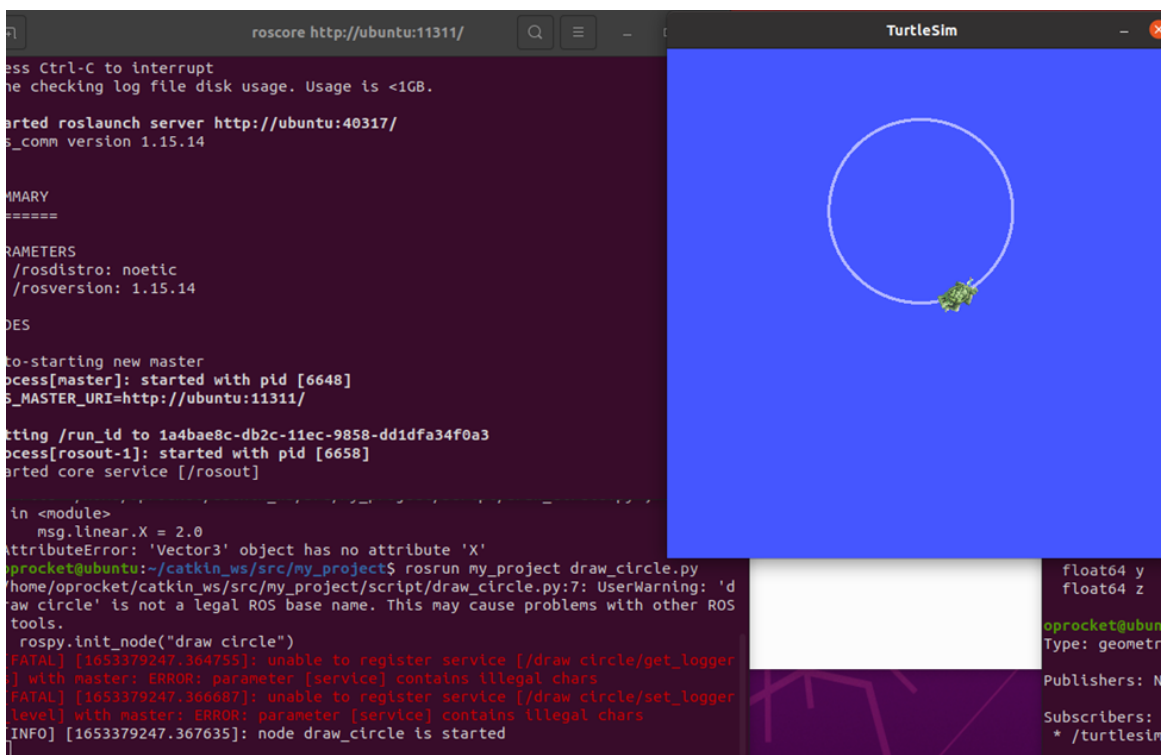
```
import queue
import rospy
from geometry_msgs.msg import Twist

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 Publish ค่าออกไป

หลังจากนั้นเราก็เซฟไว้แล้วรันไฟล์.pyที่เราเขียนไว้ลงในterminal แล้ว
หลังจากนั้นcodeที่เราเขียนก็จะ Publisher ไปสู่ turtle1/cmd_velเพื่อทำการ
เคลื่อนที่ตามโค้ดที่เขียน ดังรูปด้านล่าง

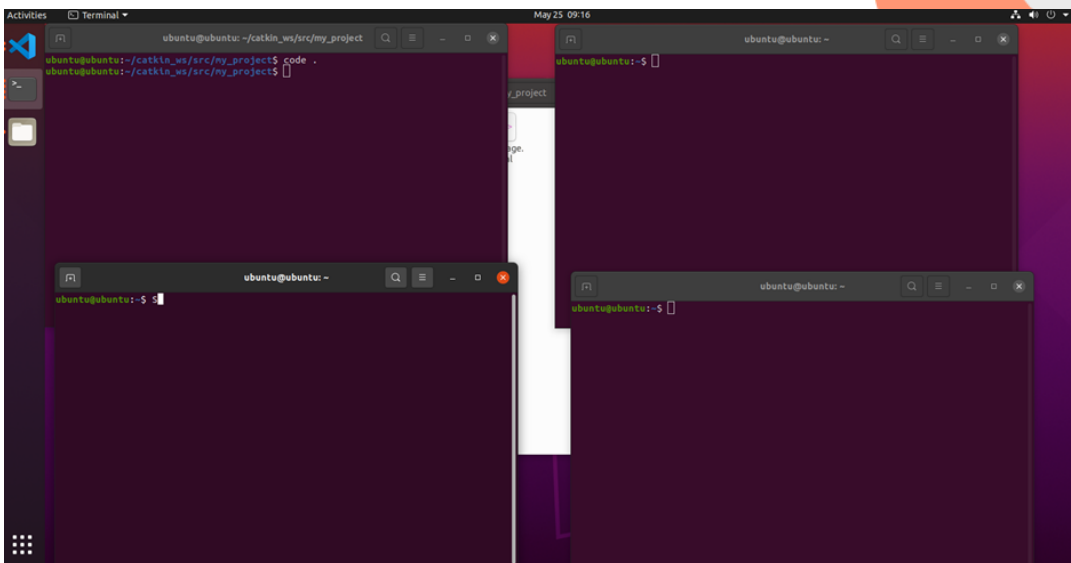


ผลลัพธ์ของโปรแกรม

บทที่ 3

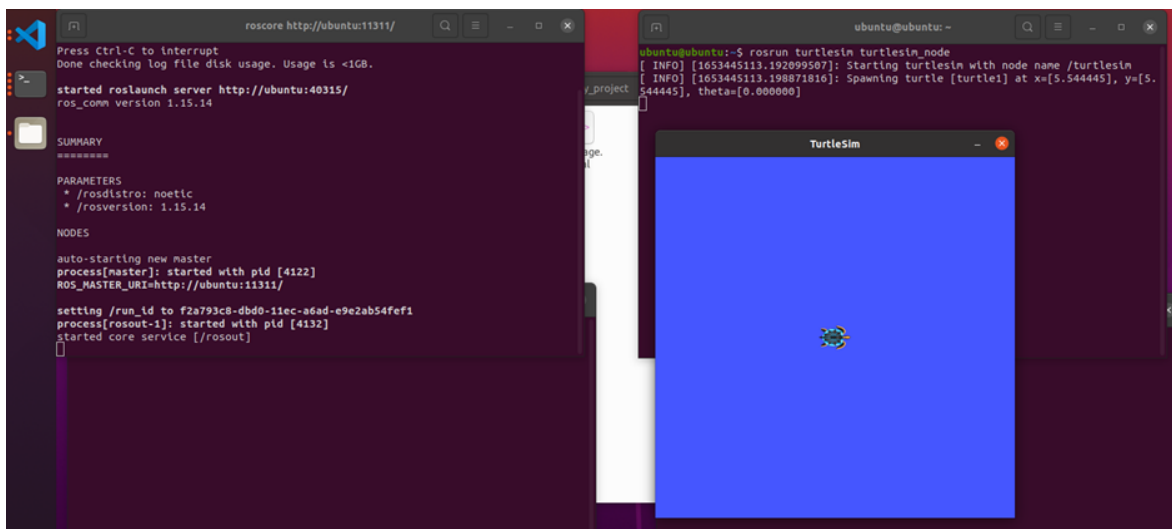
การทำ database cloud ด้กับ turtlesim และการสร้าง GUI

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



รูปการเปิด terminal ขึ้นมา 4 หน้าต่าง

Run คำสั่ง roscore อีกหน้าต่าง run คำสั่ง rosruntime turtlesim turtlesim_node



ผลลัพธ์เมื่อ run คำสั่ง `roslaunch turtlesim turtlesim_node`

Run คำสั่ง `rostopic` เพื่อดู topic ของ turtlesim

```
ubuntu@ubuntu:~$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
ubuntu@ubuntu:~$ rostopic info /turtle1/pose
Type: turtlesim/Pose

Publishers:
 * /turtlesim (http://ubuntu:43707/)

Subscribers: None

ubuntu@ubuntu:~$ S
```

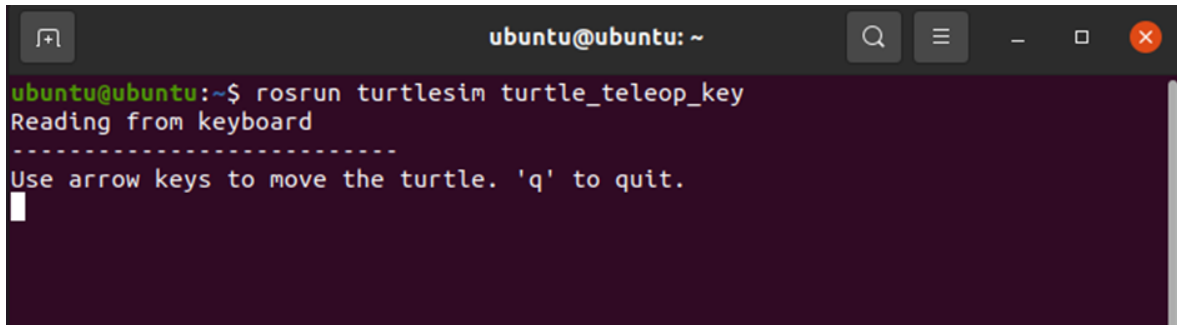
ภาพการ run คำสั่ง `rostopic`

Run คำสั่ง `rostopic echo /turtle1/pose` เพื่อดูว่ามีค่าอะไรบ้างที่ส่งมาจาก topic นี้

```
ubuntu@ubuntu: ~
ubuntu@ubuntu:~$ rostopic echo /turtle1/pose
x: 5.544444561004639
y: 5.544444561004639
theta: 0.0
linear_velocity: 0.0
angular_velocity: 0.0
---
x: 5.544444561004639
y: 5.544444561004639
theta: 0.0
linear_velocity: 0.0
angular_velocity: 0.0
---
x: 5.544444561004639
y: 5.544444561004639
theta: 0.0
linear_velocity: 0.0
angular_velocity: 0.0
---
x: 5.544444561004639
y: 5.544444561004639
theta: 0.0
```

ผลลัพธ์เมื่อ run คำสั่ง `rostopic echo /turtle1/pose`

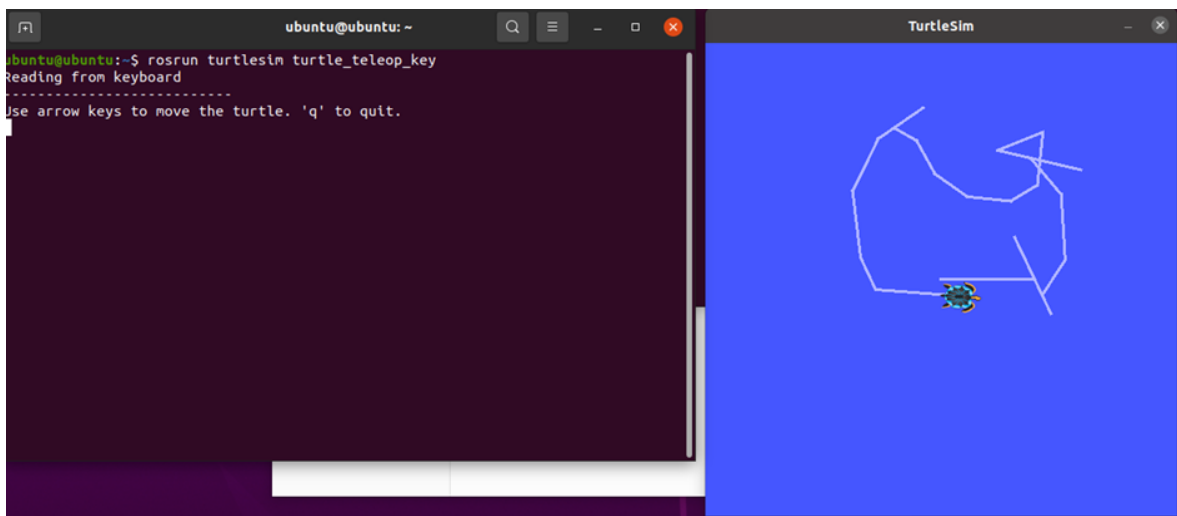
ขณะที่กำลัง run คำสั่ง rostopic echo/turtle1/pose อีก terminal ให้ run คำสั่ง
roslaunch turtlesim turtle_teleop_key



```
ubuntu@ubuntu: ~  
ubuntu@ubuntu:~$ roslaunch turtlesim turtle_teleop_key  
Reading from keyboard  
-----  
Use arrow keys to move the turtle. 'q' to quit.  
█
```

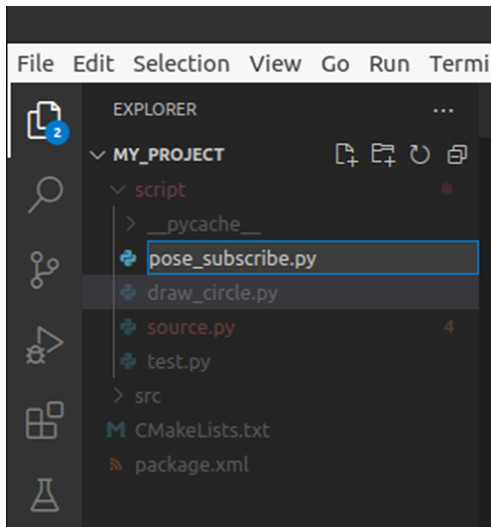
อีก terminal run คำสั่ง roslaunch turtlesim turtle_teleop_key

เมื่อขยับลูกศรบนkeyboard จะสามารถขยับทิศทางของเต่าในทิศทางที่ต้องการ
ได้



ผลลัพธ์เมื่อขยับและ run program

สร้างไฟล์ที่ชื่อ pose_subscribe.py ในไฟล์



ภาพการสร้างไฟล์ที่ชื่อ pose_subscribe.py

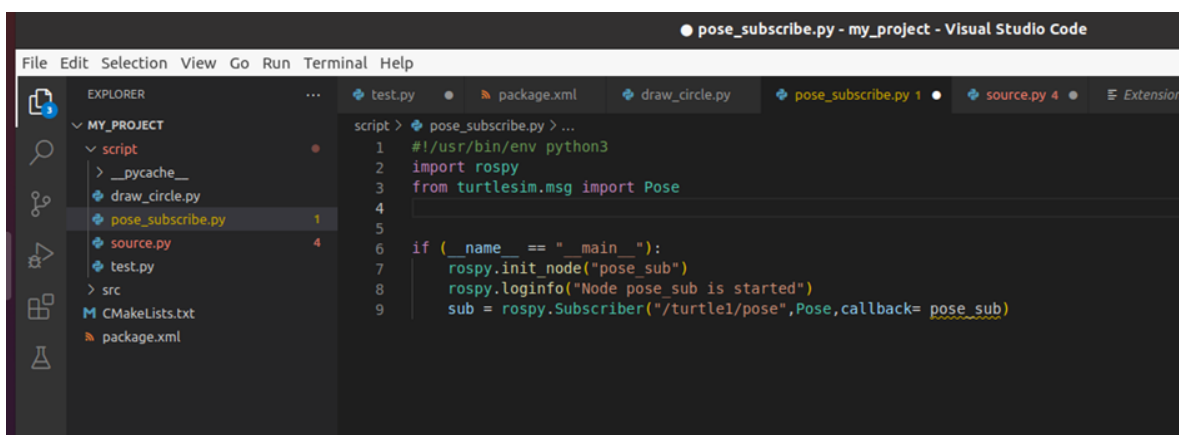
ไปที่ ที่อยู่ของ ไฟล์ pose_subscribe.py เพื่อให้ไฟล์ executable ได้โดยใช้

คำสั่ง `chmod +x pose_subscribe.py`

```
ubuntu@ubuntu:~/catkin_ws/src/my_project/script$ chmod +x pose_subscribe.py
ubuntu@ubuntu:~/catkin_ws/src/my_project/script$
```

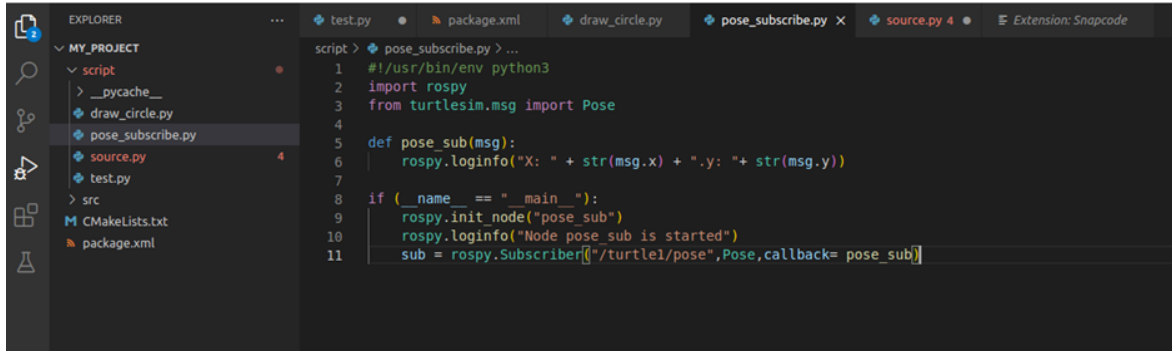
คำสั่งที่ทำให้ไฟล์ executable ได้

ทำการสร้าง node ที่เอาไว้ subscribe ข้อมูล จาก /turtle1/pose



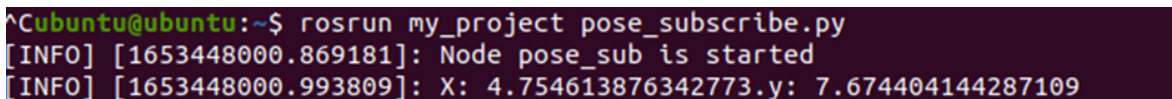
โค้ดการสร้าง node เพื่อมารับค่าจาก node /turtle1/pose

ทำการสร้างfunction pose_sub เพื่อที่จะนำค่าที่ได้รับจาก callback=pose_sub ไปแสดงค่า



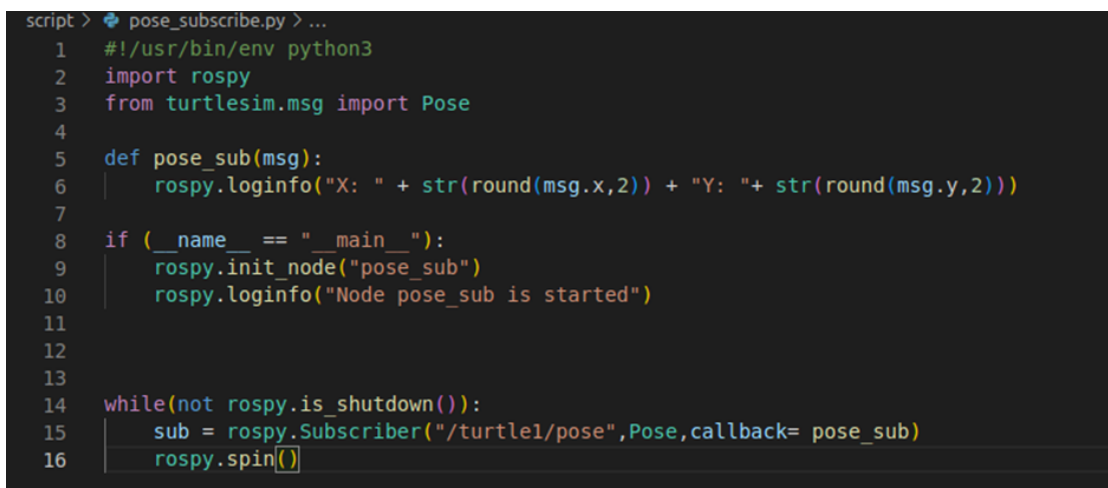
ภาพการสร้างฟังก์ชันขึ้นมาเพื่อรับค่าไปแสดงผลต่อ

Run โค้ดใน terminal เพื่อดูค่าที่ได้ subscribe มา



ผลลัพธ์ของโปรแกรม

เนื่องจากค่าที่ออกมาเป็นทศนิยมหลายตำแหน่งเราจึงใช้คำสั่ง round เพื่อปัดทศนิยมขึ้นเหลือ สองตำแหน่ง



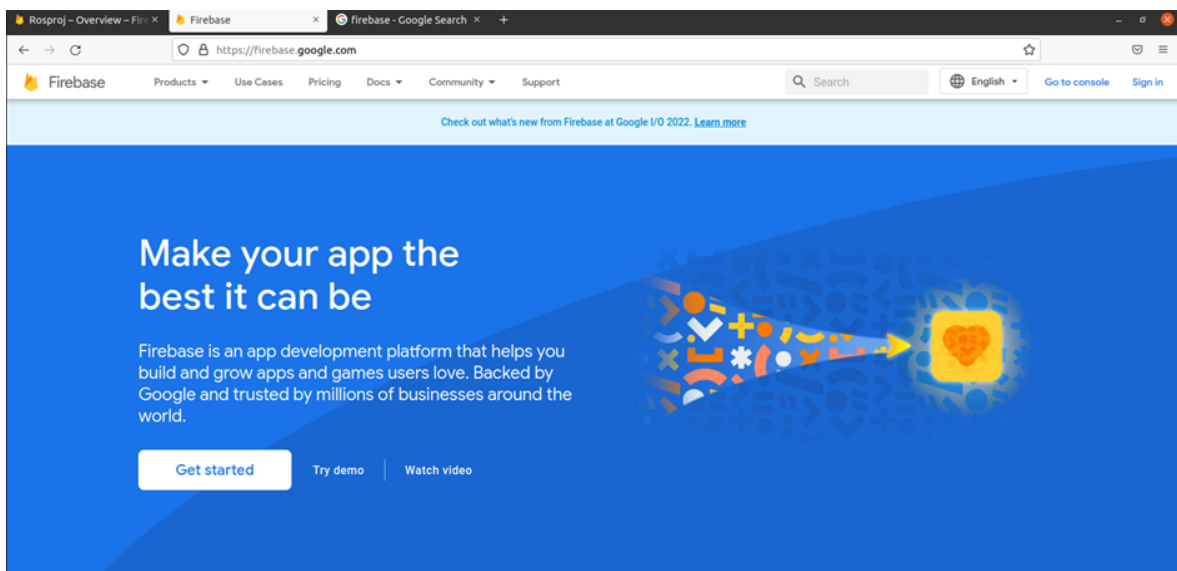
ภาพโค้ดใส่คำสั่ง round

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
```

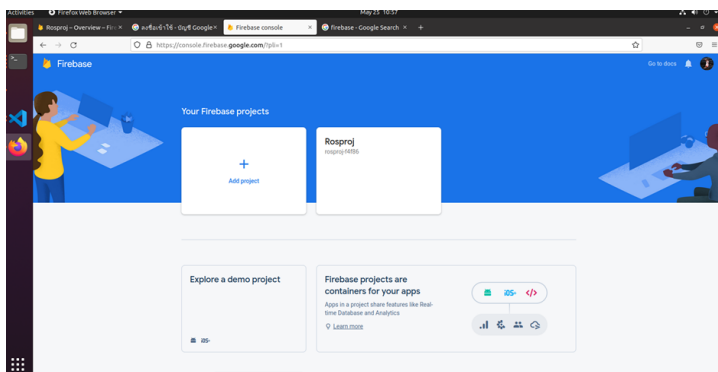
ผลลัพธ์เมื่อใส่คำสั่ง round

ทำการสมัคร firebase ใน ubuntu

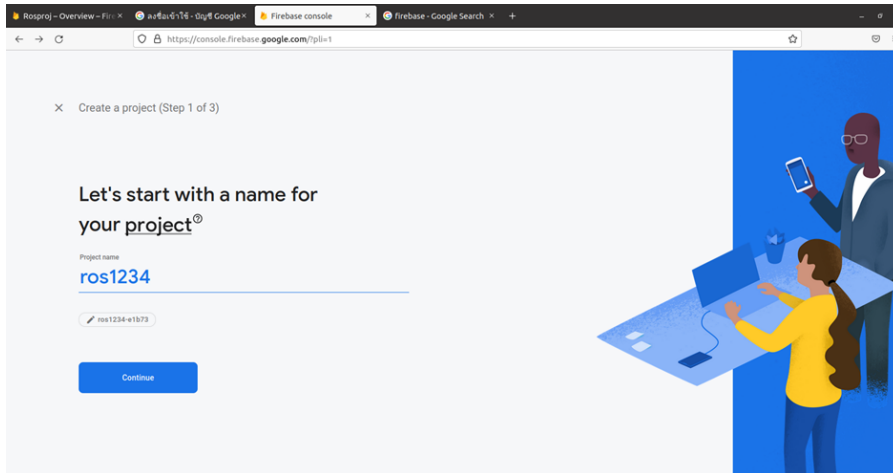


หน้าเว็บ firebase

เมื่อสมัครเรียบร้อยแล้วให้ กด get started และ สร้าง project ขึ้นมา

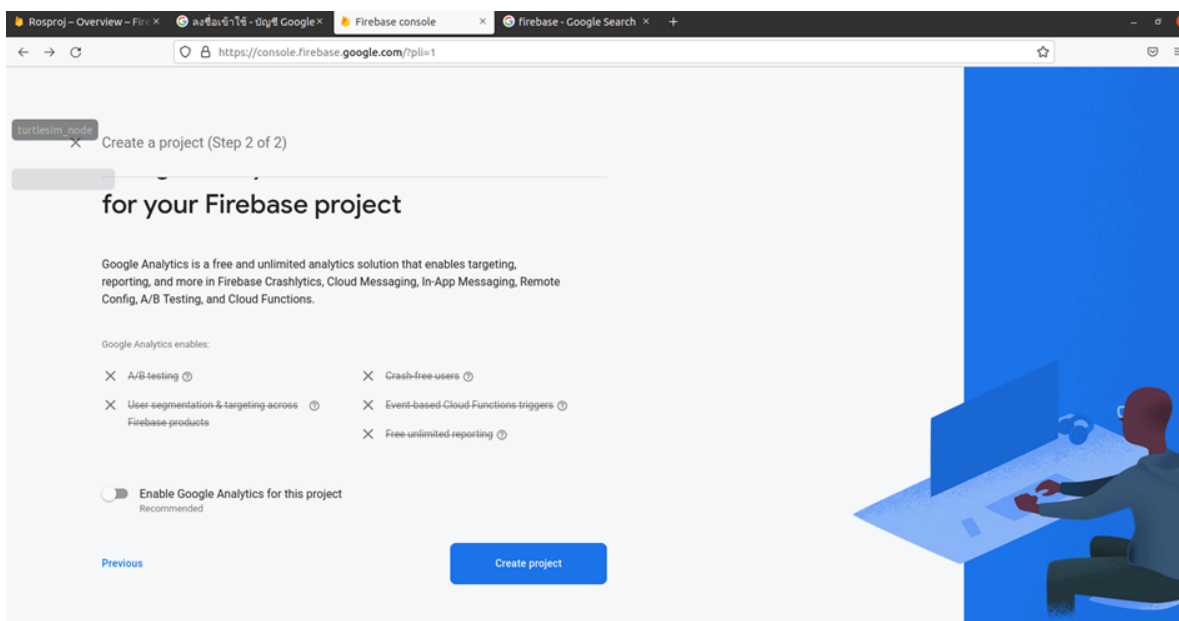


ตั้งชื่อ project

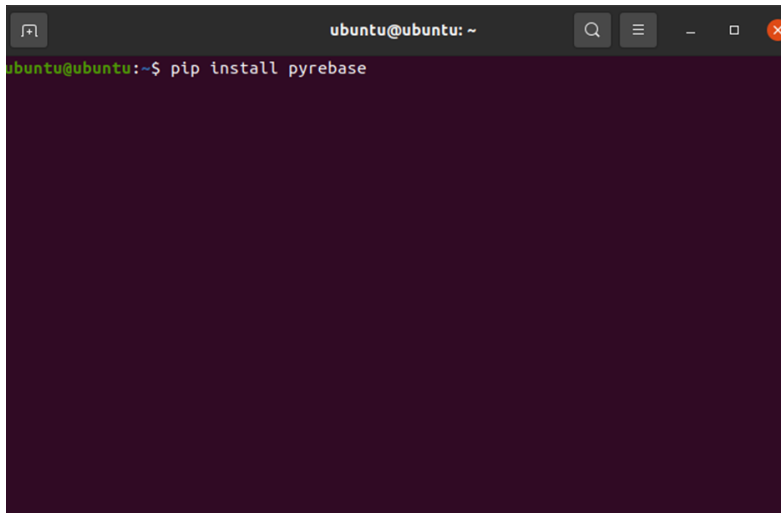


หน้าการตั้งชื่อ Project

กดยอมรับเงื่อนไข แต่ไม่กดใช้ google analytics และกด create project



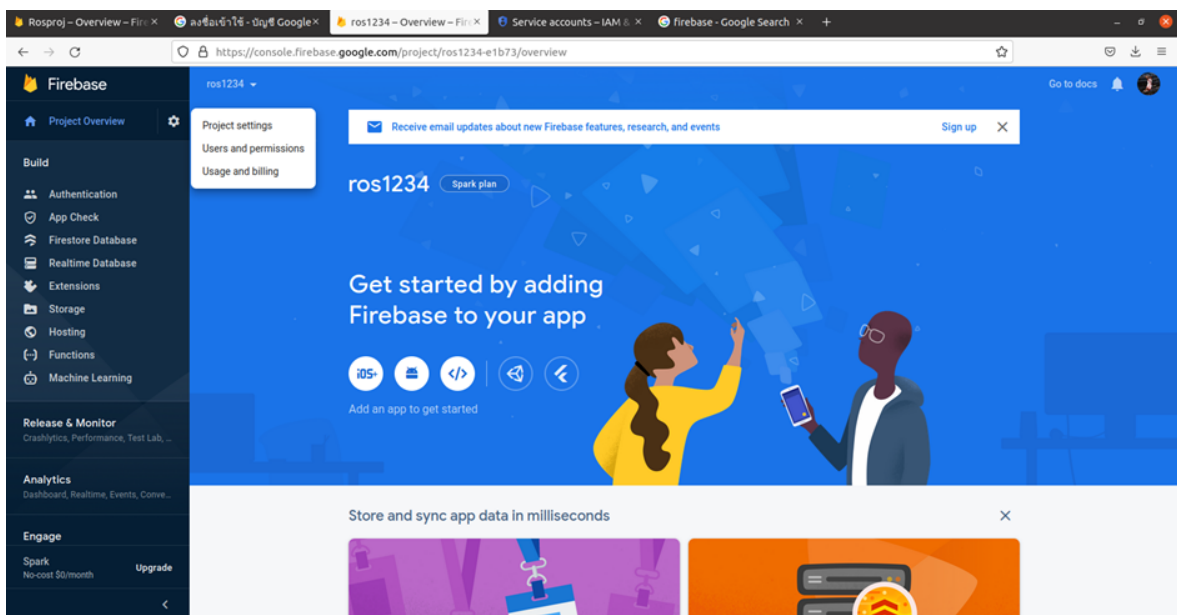
ทำการติดตั้ง library pyrebase เพื่อที่จะเชื่อมต่อกับ firebase ได้ โดยใช้คำสั่ง
pip install pyrebase ในการติดตั้ง



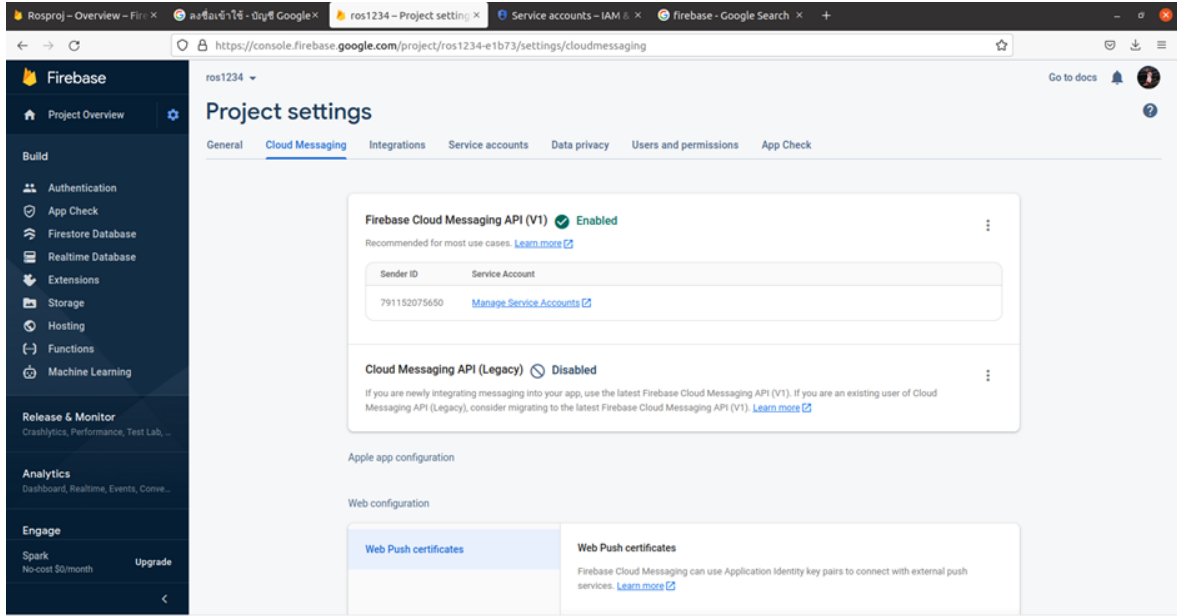
```
ubuntu@ubuntu: ~  
ubuntu@ubuntu:~$ pip install pyrebase
```

คำสั่งในการติดตั้ง pyrebase

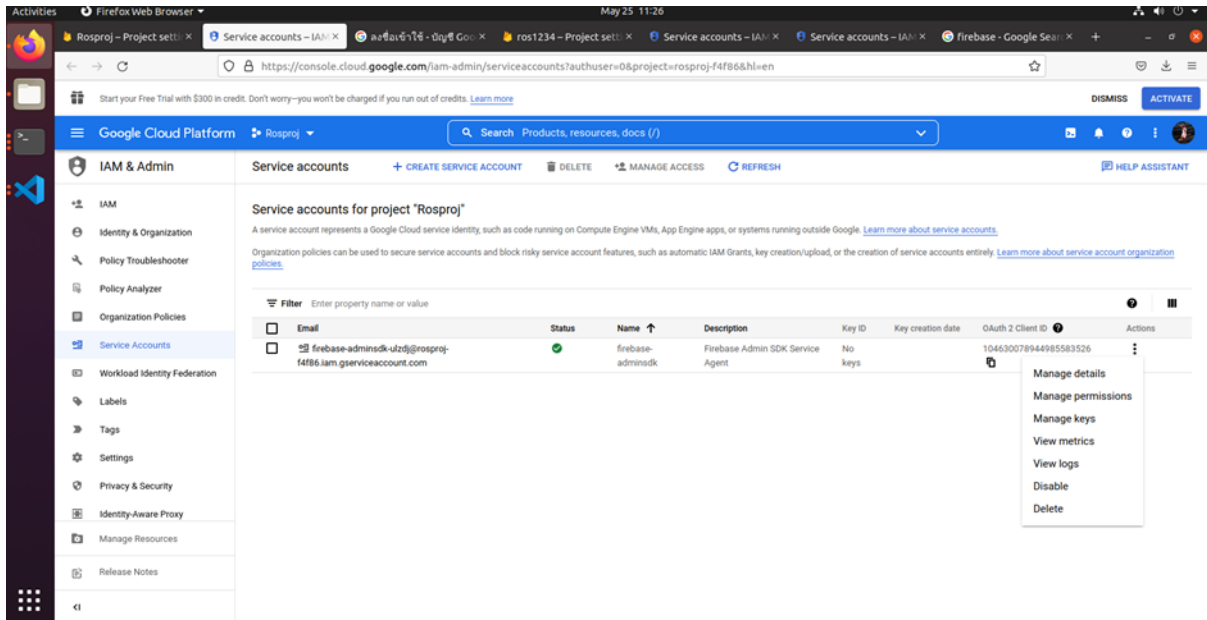
กดรูปฟันเฟืองด้านข้าง project overview และกด project setting



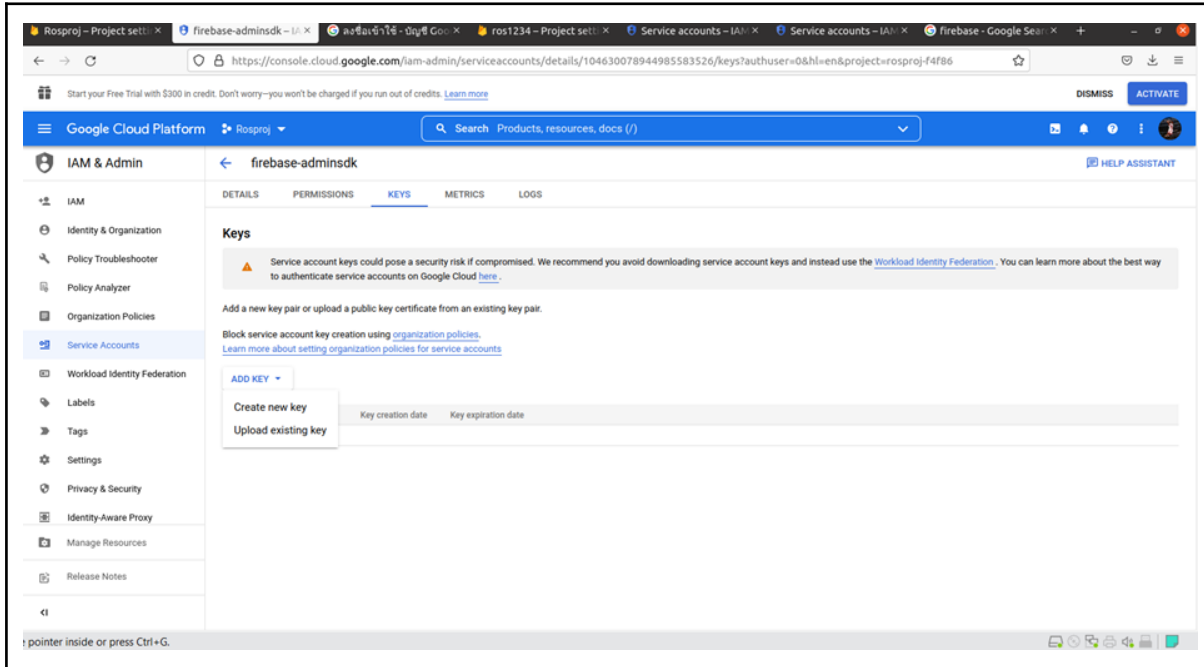
กดไปที่ cloud messaging และกดที่ Manage Service Accounts



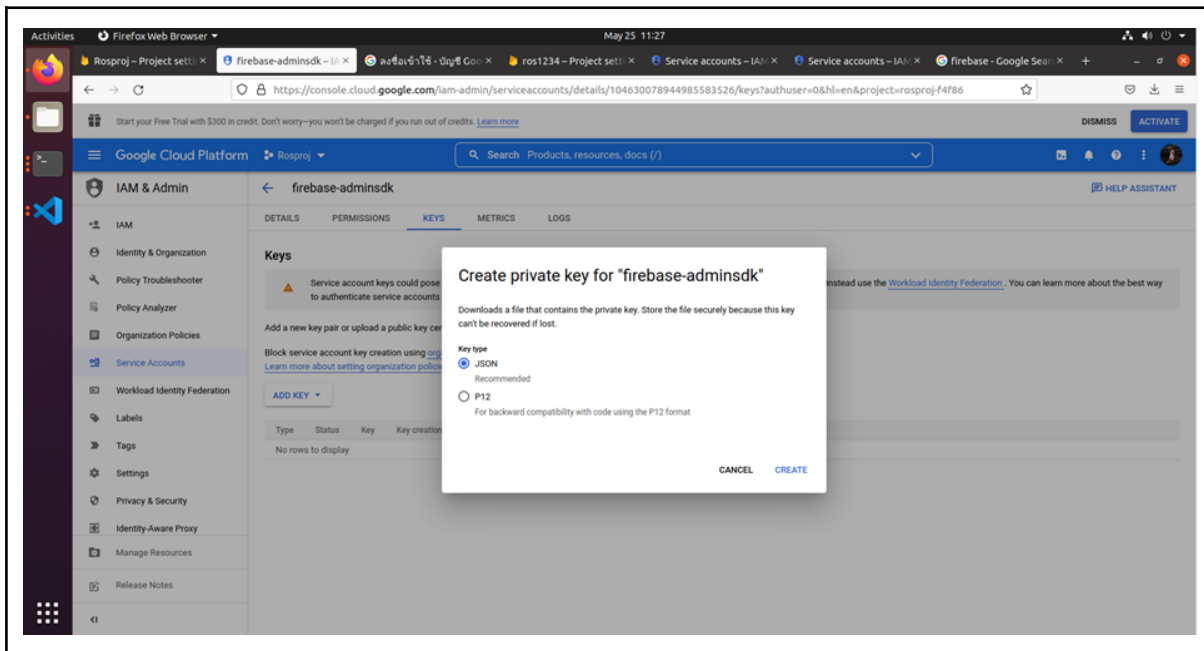
ให้กดจุดสามจุดที่ action และกด manage key



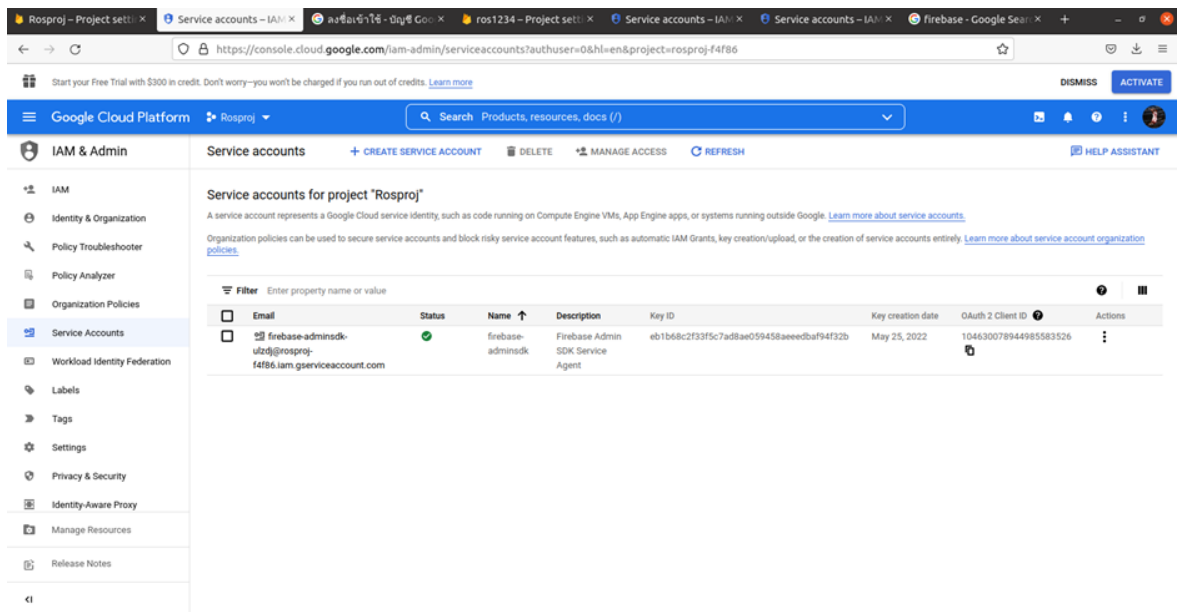
រក add key រន្ទេ create new key



រើស រើស Json រន្ទេ create

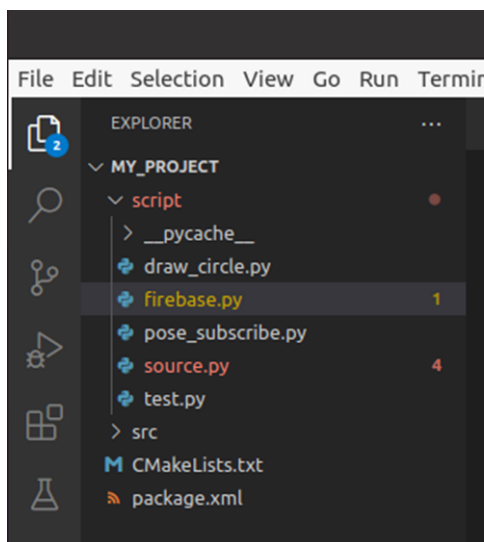


เมื่อกดสร้างkey เสร็จแล้ว กด service account จะเห็น key มา

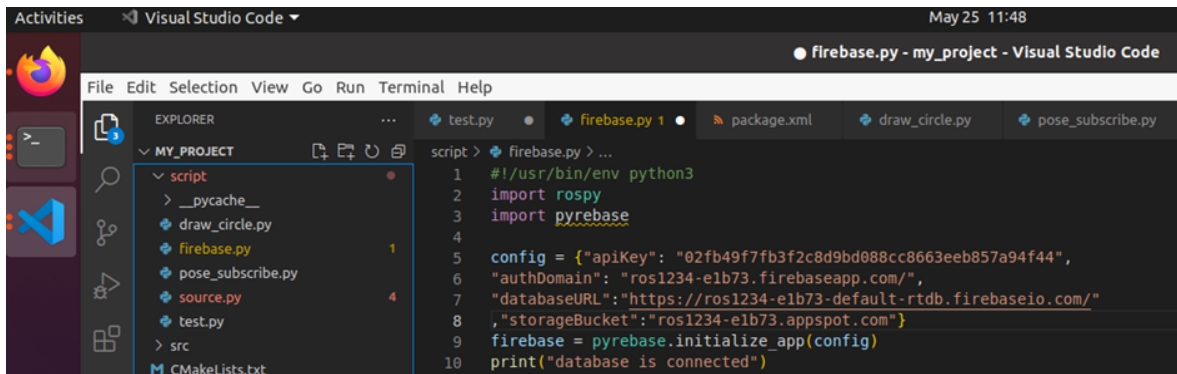


รูปkey

ให้เปิด vscode ขึ้นมาและสร้างไฟล์ firebase.py ขึ้นมาในFolder script

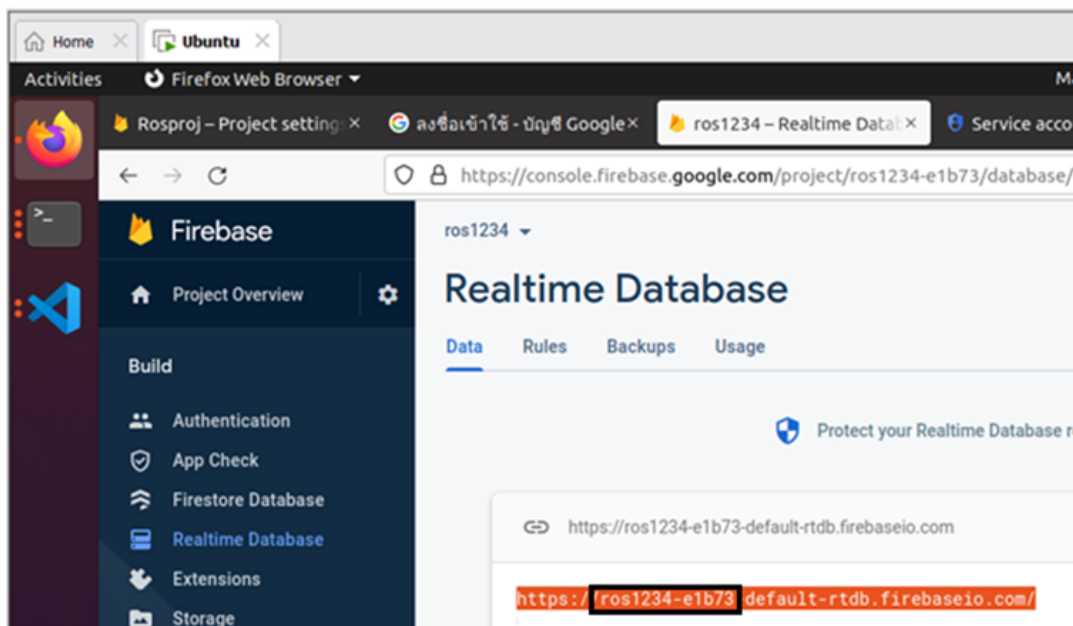


และเขียนโค้ด ดึงภาพด้านล่าง และcopy key จาก service account มาใส่ที่ apikey



```
1 #!/usr/bin/env python3
2 import rospy
3 import pyrebase
4
5 config = {"apiKey": "02fb49f7fb3f2c8d9bd088cc8663eeb857a94f44",
6 "authDomain": "ros1234-e1b73.firebaseio.com/",
7 "databaseURL": "https://ros1234-e1b73-default-rtdb.firebaseio.com/"
8 , "storageBucket": "ros1234-e1b73.appspot.com"}
9 firebase = pyrebase.initialize_app(config)
10 print("database is connected")
```

ในส่วนของ authDomain ให้copy มาจาก Realtime databaseเฉพาะส่วนที่ ตีกรอบไว้



authDomain ของ แต่ละคน จะไม่เหมือนกัน

และเติม .firebaseapp.com/

```
"authDomain": "ros1234-e1b73.firebaseio.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

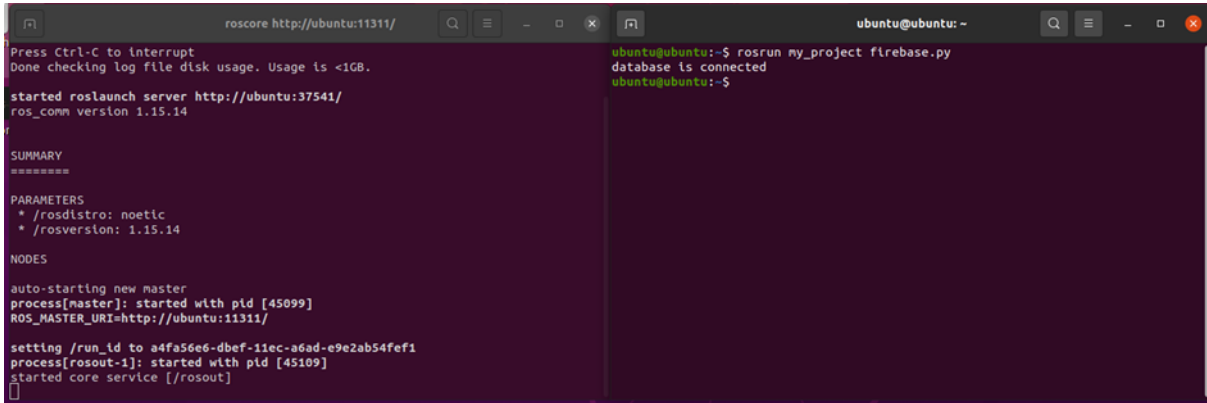
```
#!/usr/bin/env python3
import rospy
import pyrebase

config = {"apiKey":
"02fb49f7fb3f2c8d9bd088cc8663eeb857a94f44",
"authDomain": "ros1234-e1b73.firebaseio.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



```
roscore http://ubuntu:11311/
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:37541/
ros_comm version 1.15.14

SUMMARY
=====
PARAMETERS
* /rostdistro: noetic
* /rosverion: 1.15.14

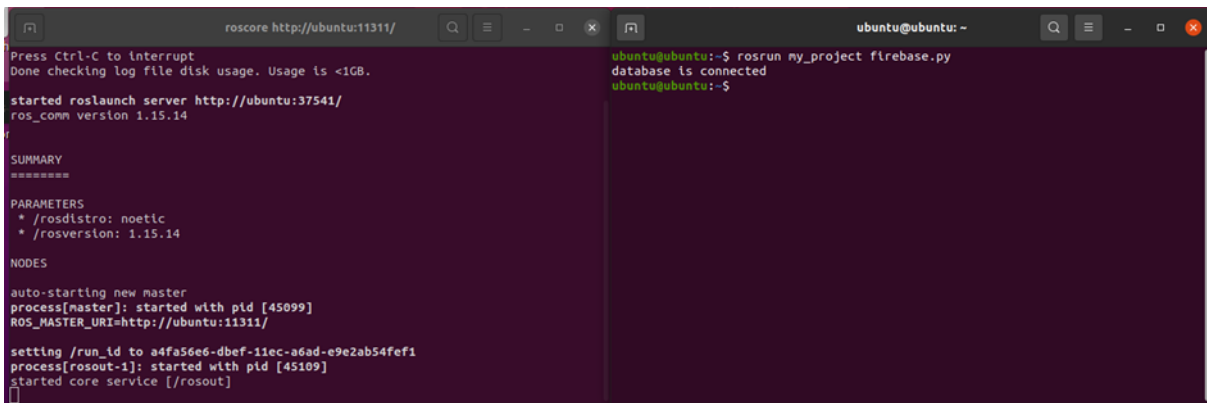
NODES

auto-starting new master
process[master]: started with pid [45099]
ROS_MASTER_URI=http://ubuntu:11311/

setting /run_id to a4fa56e6-dbef-11ec-a6ad-e9e2ab54fef1
process[rosout-1]: started with pid [45109]
started core service [/rosout]
[]

ubuntu@ubuntu:~$ rosrn my_project firebase.py
database is connected
ubuntu@ubuntu:~$
```

จากนั้นให้ไปที่firebase เพื่อตรวจสอบว่าข้อมูลเราได้ถูกส่งไปยัง firebase หรือ
ยัง หากถูกต้องจะขึ้นดั่งภาพด้านล่าง



```
roscore http://ubuntu:11311/
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:37541/
ros_comm version 1.15.14

SUMMARY
=====
PARAMETERS
* /rostdistro: noetic
* /rosverion: 1.15.14

NODES

auto-starting new master
process[master]: started with pid [45099]
ROS_MASTER_URI=http://ubuntu:11311/

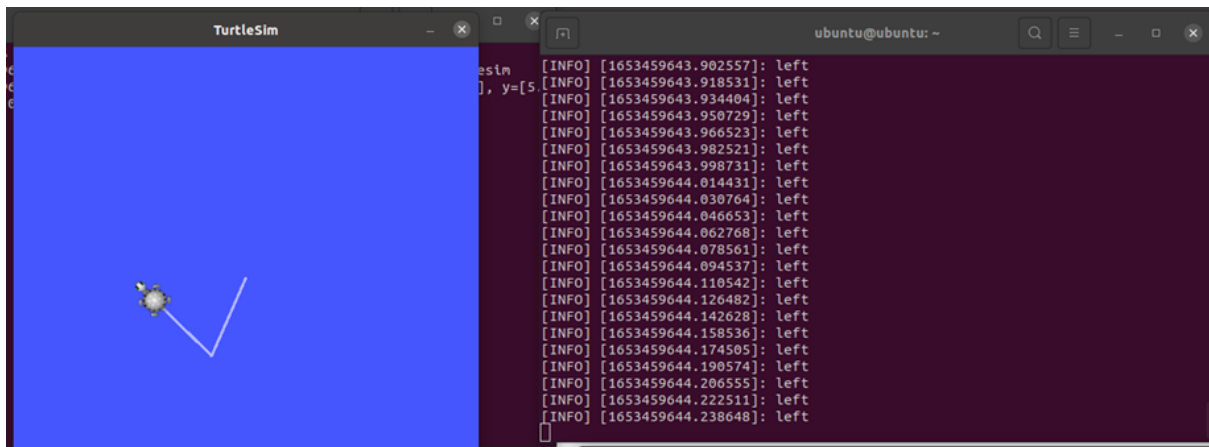
setting /run_id to a4fa56e6-dbef-11ec-a6ad-e9e2ab54fef1
process[rosout-1]: started with pid [45109]
started core service [/rosout]
[]

ubuntu@ubuntu:~$ rosrn my_project firebase.py
database is connected
ubuntu@ubuntu:~$
```

ทำการสร้างไฟล์ขึ้นมา 1 ไฟล์ ชื่อไฟล์ decision.py จะเป็นการบอกตำแหน่งของเต้า

```
1  #!/usr/bin/env python3
2  from matplotlib.pyplot import tick_params
3  import rospy
4  import pyrebase
5  from turtlesim.msg import Pose
6
7  config = {"apiKey": "02fb49f7fb3f2c8d9bd088cc8663eeb857a94f44",
8  "authDomain": "ros1234-e1b73.firebaseio.com/",
9  "databaseURL": "https://ros1234-e1b73-default-rtdb.firebaseio.com/"
10 , "storageBucket": "ros1234-e1b73.appspot.com"}
11 firebase = pyrebase.initialize_app(config)
12 print("database is connected")
13 db = firebase.database()
14
15 def pose_sub(msg):
16     if(msg.x > 0 and msg.x <= 3.6):
17         rospy.loginfo("left")
18         db.child("a").update({"data" : "left"})
19     if(msg.x > 3.6 and msg.x <= 7.2):
20         rospy.loginfo("centre")
21         db.child("a").update({"data" : "centre"})
22     if(msg.x > 7.2 and msg.x <= 11):
23         rospy.loginfo("right")
24         db.child("a").update({"data" : "right"})
25
26
27 if (__name__ == "__main__"):
28     rospy.init_node("pose_sub")
29     rospy.loginfo("Node pose_sub is started")
30
31
32 while(not rospy.is_shutdown()):
33     sub = rospy.Subscriber("/turtle1/pose", Pose, callback= pose_sub)
34     rospy.spin()
```

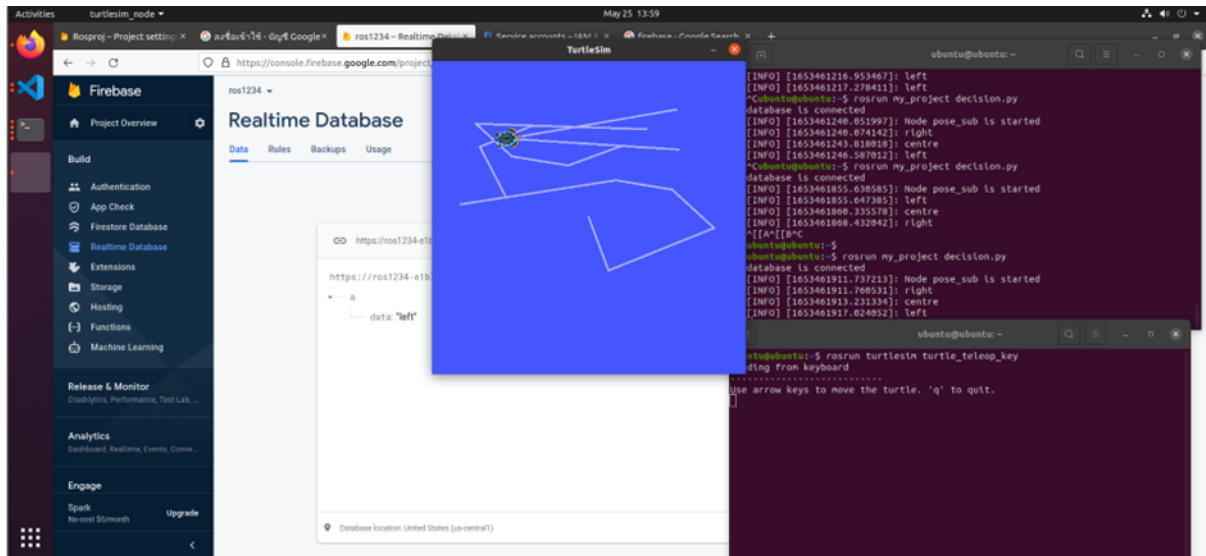
ภาพนี้เต่าอยู่ทางฝั่งซ้ายของจอ ตัวterminalด้านขวาได้บอกตำแหน่งว่าอยู่ด้านซ้าย



เราสามารถตรวจสอบว่าค่าตำแหน่งของเต่าได้ทำการ upload ไปบน firebase หรือเปล่า แต่เมื่อเปลี่ยนตำแหน่งของเต่าแล้วแต่ค่าที่แสดงมาจาก terminal ยังไม่เปลี่ยนแปลงเพราะว่าเราส่งค่าขึ้นไปเยอะเกินไป จึงต้องมีการเปลี่ยนแปลงโค้ด

```
1 #!/usr/bin/env python3
2 from matplotlib.pyplot import tick_params
3 import rospy
4 import pyrebase
5 from turtlesim.msg import Pose
6
7 config = {"apiKey": "02fb49f7fb3f2c8d9bd088cc8663eeb857a94f44",
8 "authDomain": "ros1234-e1b73.firebaseio.com/",
9 "databaseURL": "https://ros1234-e1b73-default-rtbd.firebaseio.com/"
10 , "storageBucket": "ros1234-e1b73.appspot.com"}
11 firebase = pyrebase.initialize_app(config)
12 print("database is connected")
13 db = firebase.database()
14
15 def pose_sub(msg):
16     if(msg.x > 0 and msg.x <= 3.6):
17         rospy.loginfo("left")
18         db.child("a").update({"data": "left"})
19     if(msg.x > 3.6 and msg.x <= 7.2):
20         rospy.loginfo("centre")
21         db.child("a").update({"data": "centre"})
22     if(msg.x > 7.2 and msg.x <= 11):
23         rospy.loginfo("right")
24         db.child("a").update({"data": "right"})
25
26
27 if (__name__ == "__main__"):
28     rospy.init_node("pose_sub")
29     rospy.loginfo("Node pose_sub is started")
30
31
32 while(not rospy.is_shutdown()):
33     sub = rospy.Subscriber("/turtle1/pose", Pose, callback= pose_sub)
34     rospy.spin()
35
```

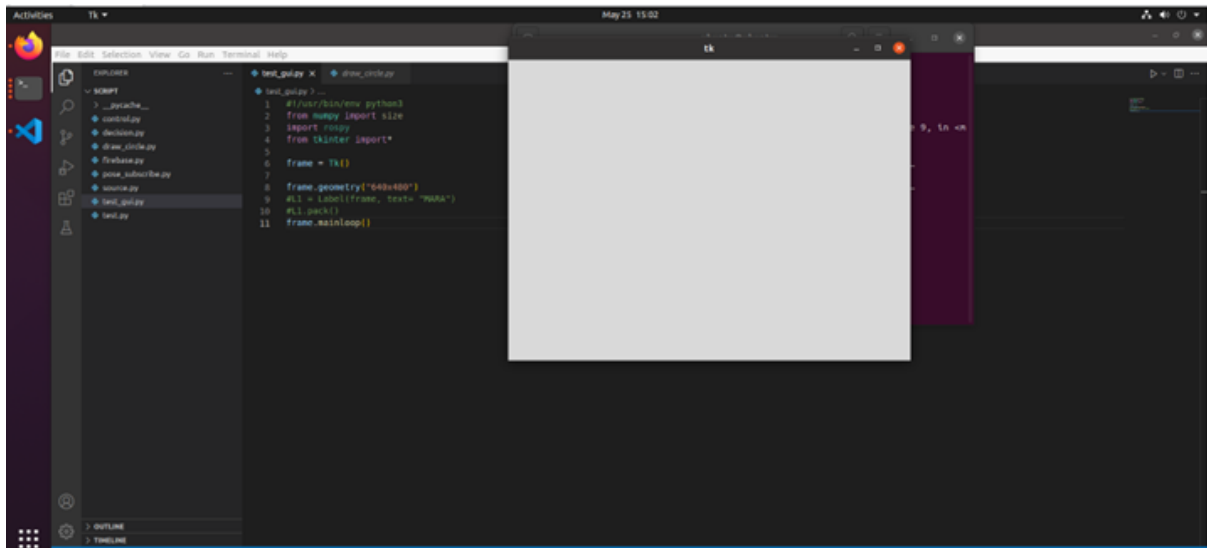
เมื่อ run โปรแกรมจะได้หน้าต่างแบบนี้



ในส่วนนี้เราจะทำการสร้างGUI ขึ้นมาโดยใช้ library import tkinter เราจะสร้างหน้าต่างขนาด 640x480

```
1 #!/usr/bin/env python3
2 from numpy import size
3 import rospy
4 from tkinter import*
5
6 frame = Tk()
7
8 frame.geometry("640x480")
9 L1 = Label(frame, text= "MARA")
10 L1.pack()
11 frame.mainloop()
12
13
```

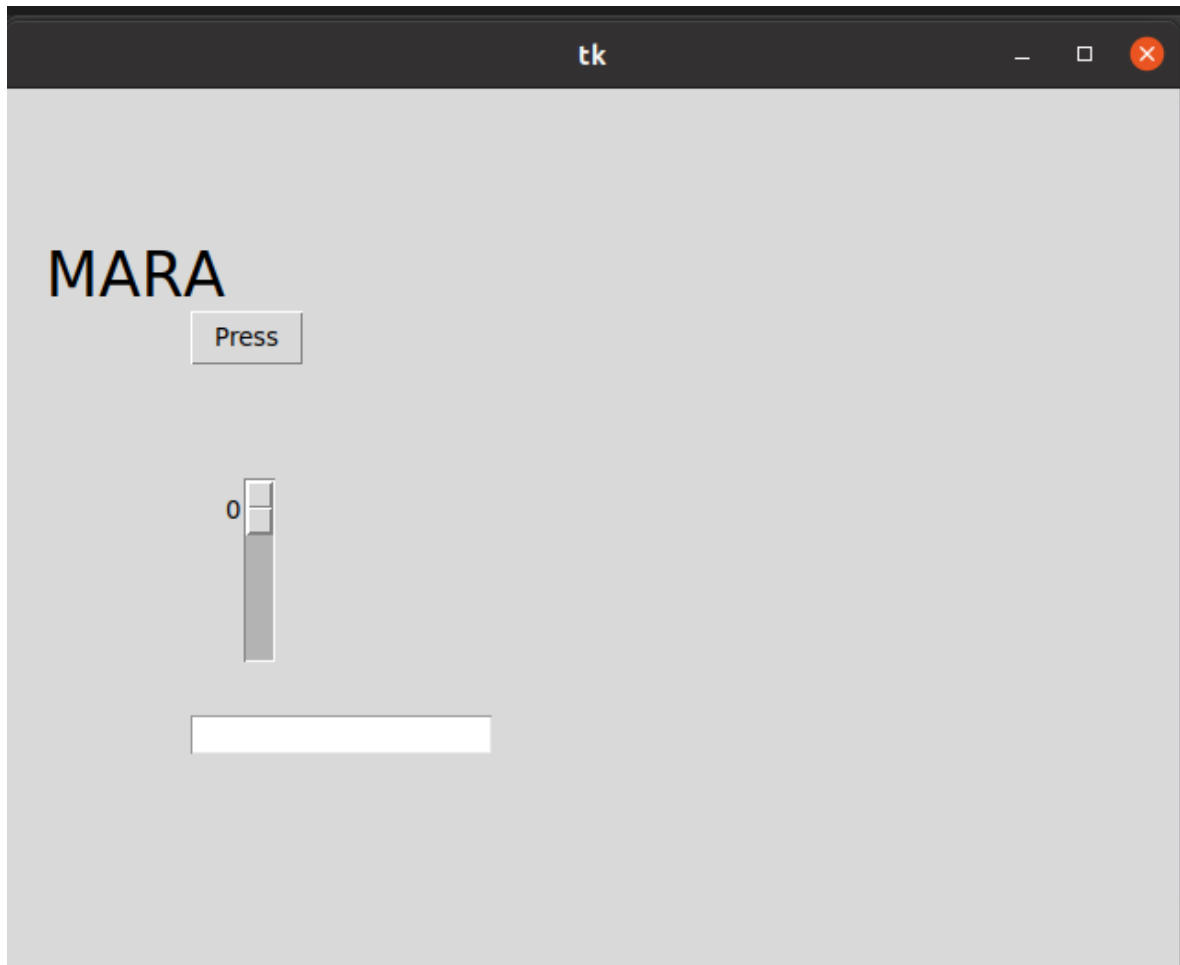
เมื่อrun โปรแกรมจะได้หน้าต่างแบบนี้



ในขั้นนี้เราได้เพิ่ม ตัวหนังสือ ปุ่ม แถบเลื่อน และช่องกรอกข้อมูล

```
1  #!/usr/bin/env python3
2  from numpy import size
3  import rospy
4  from tkinter import*
5
6  frame = Tk()
7
8  frame.geometry("640x480")
9  L1 = Label(frame, text= "MARA", font= ('Aerail',25))
10 L1.place(x=20,y=80)
11 B = Button(frame, text = "Press")
12 B.place(x=100, y= 120)
13 S = Scale(frame)
14 S.place(x=100, y=210)
15 E = Entry(frame)
16 E.place(x=100,y=340)
17
18 frame.mainloop()
```

จะได้ผลลัพธ์ดังรูป



ผลลัพธ์ของโค้ด

บทที่ 4

ใช้ arduino สื่อสารกับระบบ ROS และก็มีการสร้าง simulation ใน tinker cad

ต่อไปเราจะเข้าสู่ขั้นตอนของการสร้าง GUI ที่สามารถ active กับ ROS ได้ ขั้นแรกเราจะลองทำให้ function ของ button จากงานเขียนโค้ด Python ใน VScode นั้นสามารถส่ง command ด้วยคำสั่งโค้ดรูปด้านล่าง เมื่อ command ไปแล้วสั่งให้ command นั้นออกคำสั่งอะไรบางอย่างตามที่เราได้เขียนลงไป

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)
```

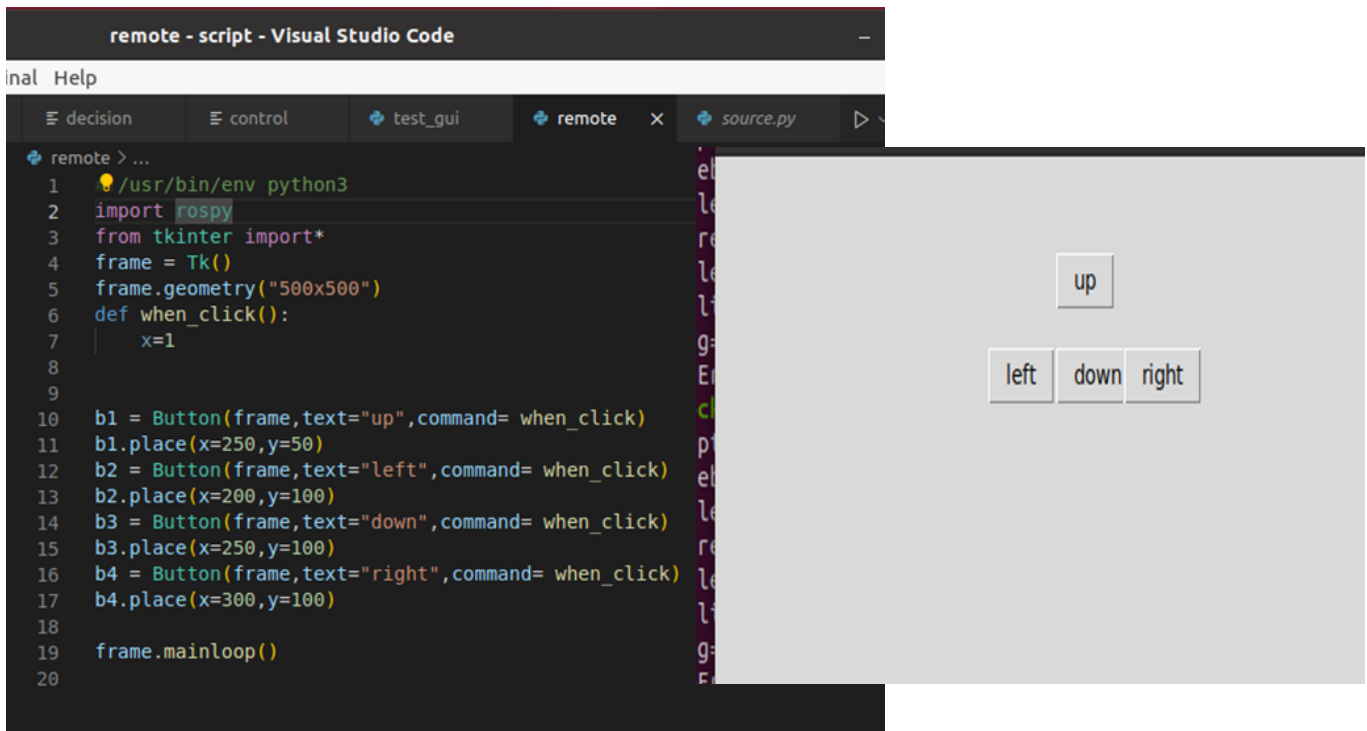
ต่อไปจะลองใช้ของตัว scale ดูว่าจะสามารถเล่นกับค่าของ scale ได้ไหม โดยการเราจะให้เมื่อเลื่อน scale แล้วจะไปคำสั่งของ Show text ตามรูปตัวอย่างด้านล่าง

```
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)
```

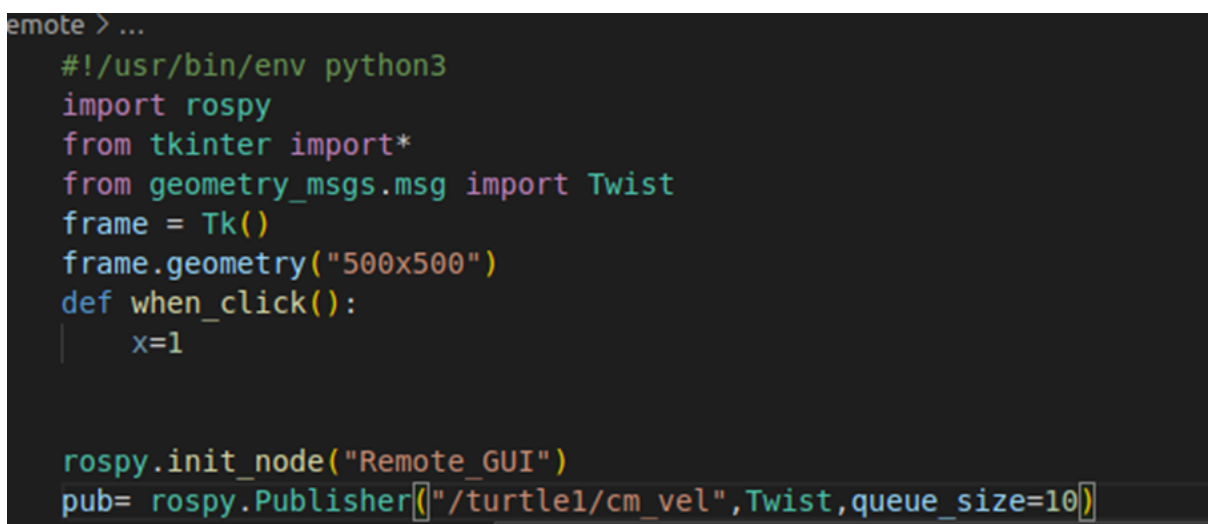


เมื่อเราเรียนรู้วิธีการเราการทำ command ของแต่ละปุ่มได้แล้วทีนี้ขั้นตอนต่อไปเราจะมาทำการลองเชื่อมโยงกับ Ros โดยตัวอย่างนี้เราจะลองสร้างปุ่มขึ้นมาเพื่อบังคับการเดินของตัวเต่าใน turtlesim ว่าจะได้หรือไม่ โดยขั้นแรกเราจะสร้างปุ่มขึ้นมาก่อนจากโค้ดที่เราเคยสร้างขึ้นมา **ตัวอย่างโค้ดด้านล่าง**



```
remote - script - Visual Studio Code
inal Help
decision control test_gui remote source.py
remote > ...
1 /usr/bin/env python3
2 import rospy
3 from tkinter import*
4 frame = Tk()
5 frame.geometry("500x500")
6 def when_click():
7     x=1
8
9
10 b1 = Button(frame,text="up",command= when_click)
11 b1.place(x=250,y=50)
12 b2 = Button(frame,text="left",command= when_click)
13 b2.place(x=200,y=100)
14 b3 = Button(frame,text="down",command= when_click)
15 b3.place(x=250,y=100)
16 b4 = Button(frame,text="right",command= when_click)
17 b4.place(x=300,y=100)
18
19 frame.mainloop()
20
```

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



```
remote > ...
#!/usr/bin/env python3
import rospy
from tkinter import*
from geometry_msgs.msg import Twist
frame = Tk()
frame.geometry("500x500")
def when_click():
    x=1

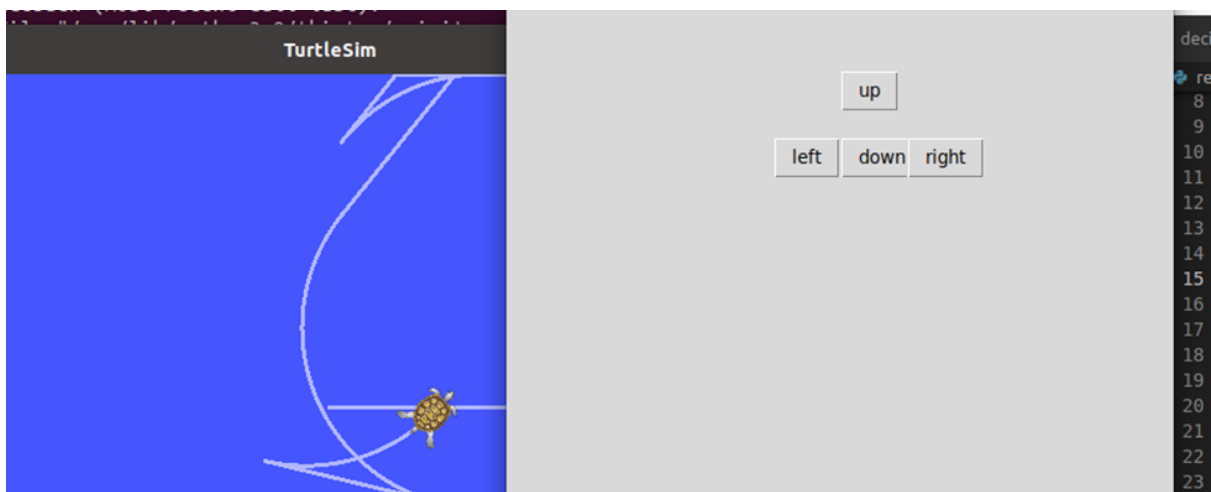
rospy.init_node("Remote_GUI")
pub= rospy.Publisher("/turtle1/cm_vel",Twist,queue_size=10)
```

จากนั้นเราจะทำการกำหนดการส่งค่าจากปุ่มไปเป็นคณหน้าดอยหลังซ้ายขวาตามปกติที่เราเคยทำไปโดยค่าความเร็วหรือมุมนั้นเราสามารถสร้างกำหนดขึ้นมาเองได้แล้วแต่ที่เราต้องการ ใค้ดตัวอย่างตามด้านล่าง

```
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)
```

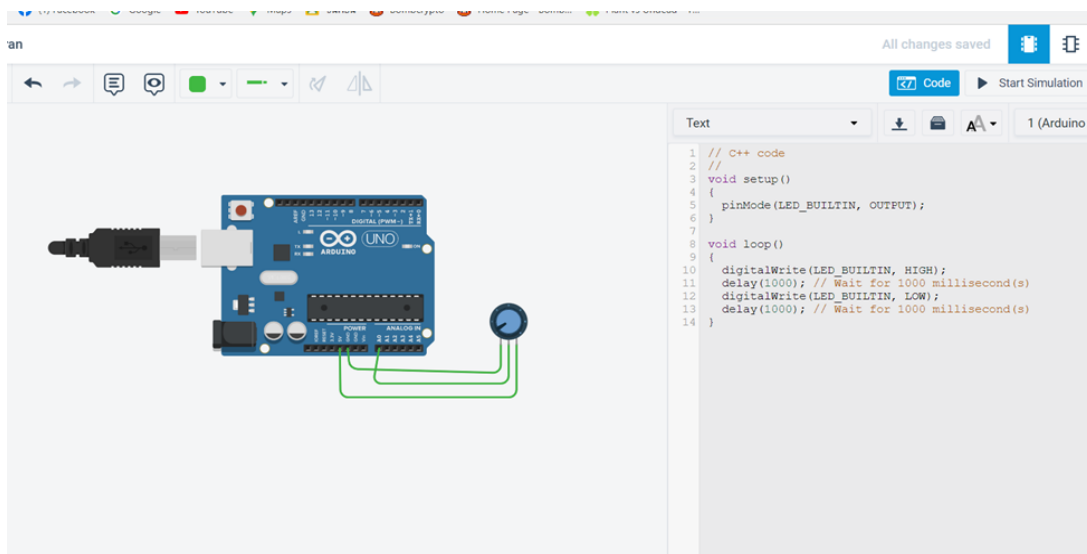
```
b1 = Button(frame,text="up",command= FW)
b1.place(x=250,y=50)
b2 = Button(frame,text="left",command= L)
b2.place(x=200,y=100)
b3 = Button(frame,text="down",command= D)
b3.place(x=250,y=100)
b4 = Button(frame,text="right",command= R)
b4.place(x=300,y=100)
```

เราก็จะได้ค่าของปุ่มที่เราสามารถส่งค่าคำสั่ง ROS ไปสู่ turtlesim ได้ เป็นการจับตัวของเรื่อง GUI.



บทต่อไป เราจะเราจะมาลองใช้การต่อวงจร Arduino simulation ของ tinkercad เราไป simulate circuit สิ่งทีเตรียมใน simulation

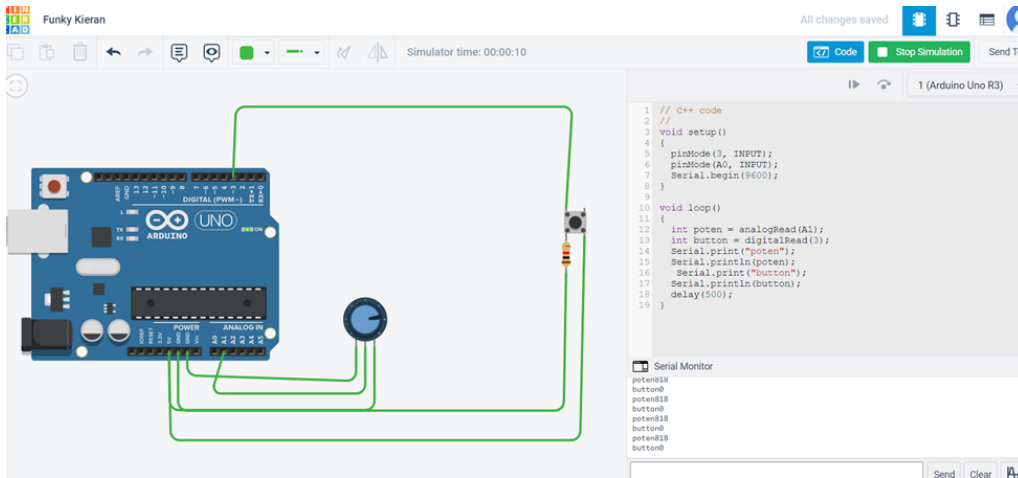
- Arduino
- Multimeter
- Potentiometer
- Button



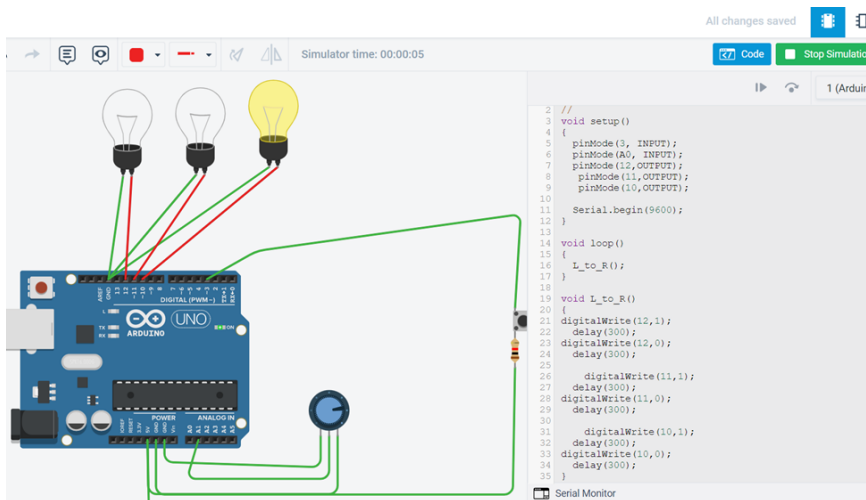
เราจะลองการใช้อ่านค่า potential จากการลองการเขียนโค้ดในตัวของ Arduino ว่าจะได้ค่ามาได้เท่าไร ตามการหมุนของ potentiometer ค่าที่อ่านได้ออกจะได้อ่านมา 0-1023 โดยค่าที่ออกมาได้จะเป็นค่าของ Analog



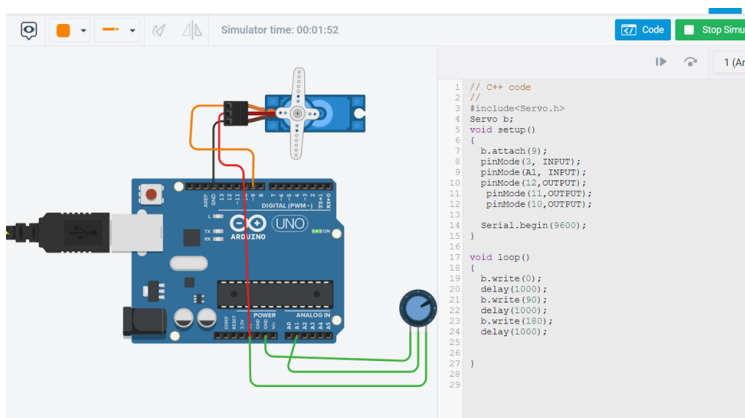
เราจะลองอ่านค่า digital ของ port 3 โดยใช้ ตัวของ button เข้ามาช่วยทำการดูค่า digital เป็น 0 กับ 1 เมื่อกด button นั้น



ต่อไปเราลองต่อ output ออกมาทำวงจร โดยลองใช้ หลอดไฟดู ทำวงจรของไฟวิ่งจากซ้ายไปขวาทำการเปิดปิดและใส่ delay จากโค้ดตัวอย่าง



ตัวอย่างการต่อ servo ของ Arduino โดยเราจะเรียกใช้ library ของ servo และเราจะลองทำการกำหนดมุมของ servo ต่างๆจากโค้ดรูปทางด้านล่าง



บทต่อไปเราจะไปลง Arduino บน platform ของ ubuntu เพื่อทำการควบคุม board arduinoของเรา โดยขั้นตอน install Arduino ขั้นแรกทำการใส่ คำสั่งตามขั้นตอนต่อไปนี้นำลงไป 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
```

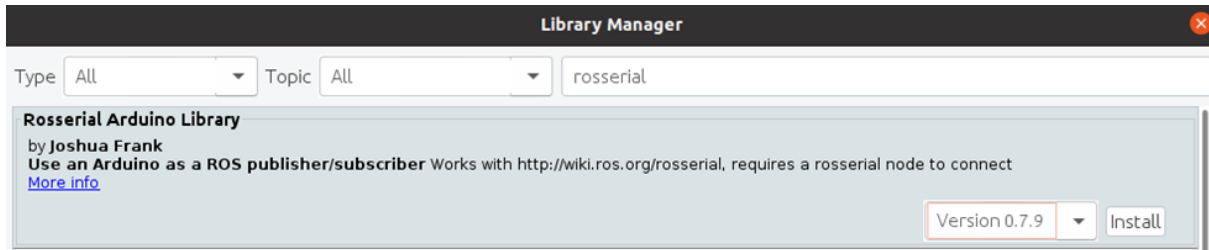
```
sudo ./install.sh
```



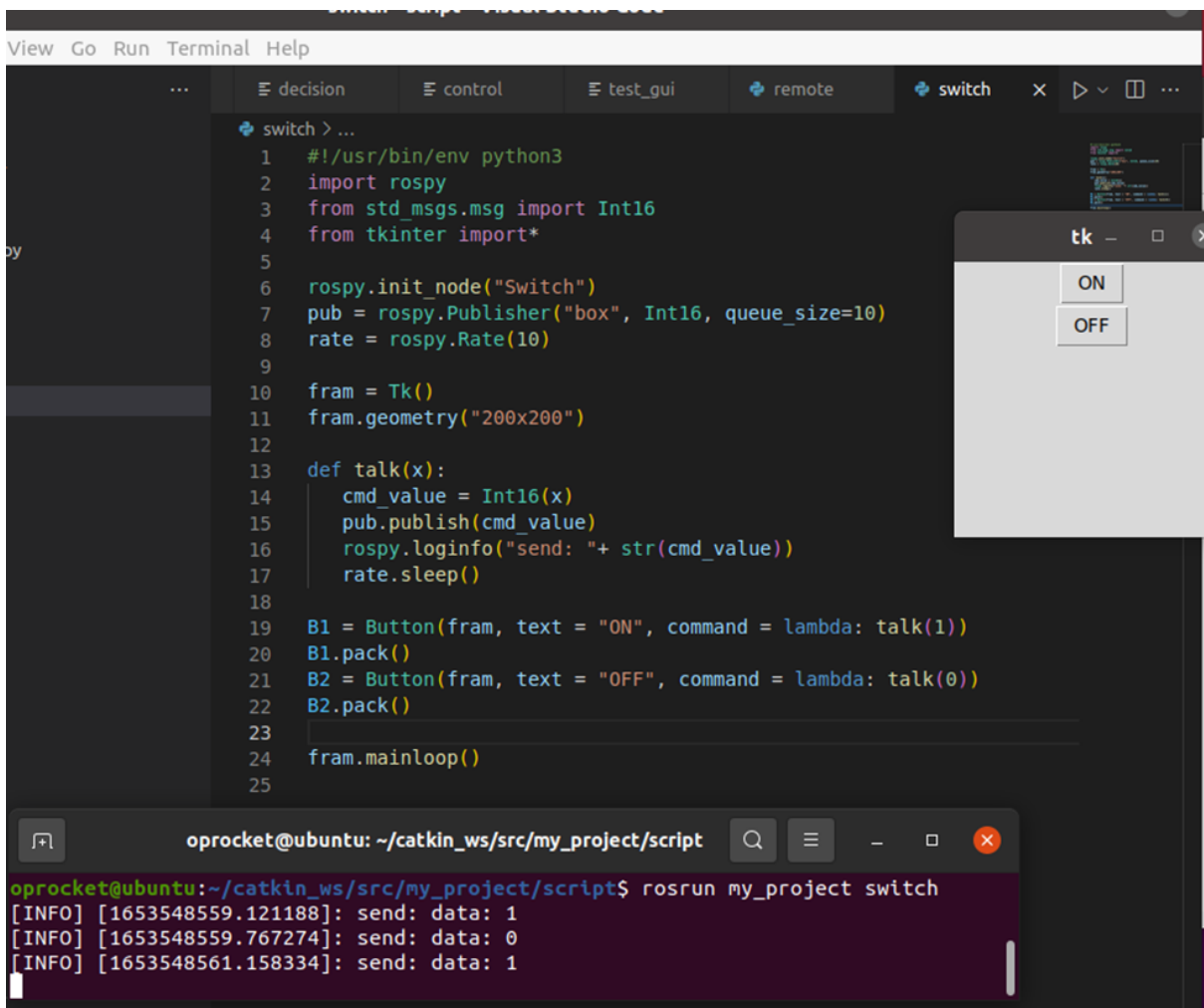
จากนั้นเมื่อเราลองต่อ board Arduino เข้ากับ pc ของเราแล้วเราจะทำการ เช็ทการ เชื่อด้วยคำสั่ง dmesg ลงใน terminal ถ้าเชื่อมต่อสำเร็จจะขึ้นตามหน้า terminal ตามรูปด้านล่าง

```
[ 1824.273815] hub_event+0x152/0x3b0
[ 1824.273817] process_one_work+0x220/0x3c0
[ 1824.273839] worker_thread+0x4d/0x3f0
[ 1824.273841] ? process_one_work+0x3c0/0x3c0
[ 1824.273842] kthread+0x12b/0x150
[ 1824.273844] ? set_kthread_struct+0x40/0x40
[ 1824.273864] ret_from_fork+0x22/0x30
[ 1824.273868] </TASK>
[ 1824.273869] ---[ end trace 73a3823684a25fed ]---
[ 1824.281552] usb 2-2.2: ch34x converter now attached to ttyUSB0
[ 1824.489071] usbcore: registered new interface driver ch341
[ 1824.489096] usbserial: USB Serial support registered for ch341-uart
[ 1866.450292] usb 2-2.2: USB disconnect, device number 5
[ 1866.450617] ch34x ttyUSB0: ch34x converter now disconnected from ttyUSB0
[ 1866.450663] ch34x 2-2.2:1.0: device disconnected
[ 1871.493719] usb 2-2.2: new full-speed USB device number 6 using uhci_hcd
[ 1871.801946] usb 2-2.2: New USB device found, idVendor=1a86, idProduct=7523, b
cdDevice= 2.64
[ 1871.801952] usb 2-2.2: New USB device strings: Mfr=0, Product=2, SerialNumber
=0
[ 1871.801953] usb 2-2.2: Product: USB Serial
[ 1871.805512] ch34x 2-2.2:1.0: ch34x converter detected
[ 1871.814033] usb 2-2.2: ch34x converter now attached to ttyUSB0
oprocket@ubuntu:~/CH340_Source/CH341SER$
```

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



เมื่อเราได้ library แล้วเราจะลองเขียนโปรแกรม เอาไว้ควบคุม จากการสร้าง GUI ครั้งก่อน จากตัวของ virtual studio โดยตัวอย่างโค้ดที่เราจะใช้ ชั้นแรกเราจะลองทำตัวของปุ่มเป็น publisher ส่งค่าออกไปได้จากการกดปุ่มด้านล่าง



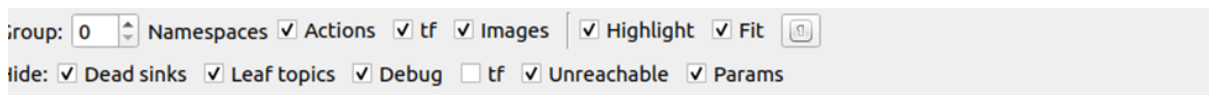
ต่อไปเราจะสร้างอีก 1 node เป็น node ของตัว Arduino โดย node นี้ต้องเชื่อมกับ node switch ของเราที่เราสร้างขึ้นมาพร้อมกับตัว GUI โดยการสร้าง node ในตัวของ Arduino และเป็นตัว subscribe ตัวของเราสร้างไว้ตามตัวเอง publisher ของเราชื่อ box เราก็จะ subscribe มาจากโค้ดตามตัวอย่างด้านล่าง

```
onoffnode 5
#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



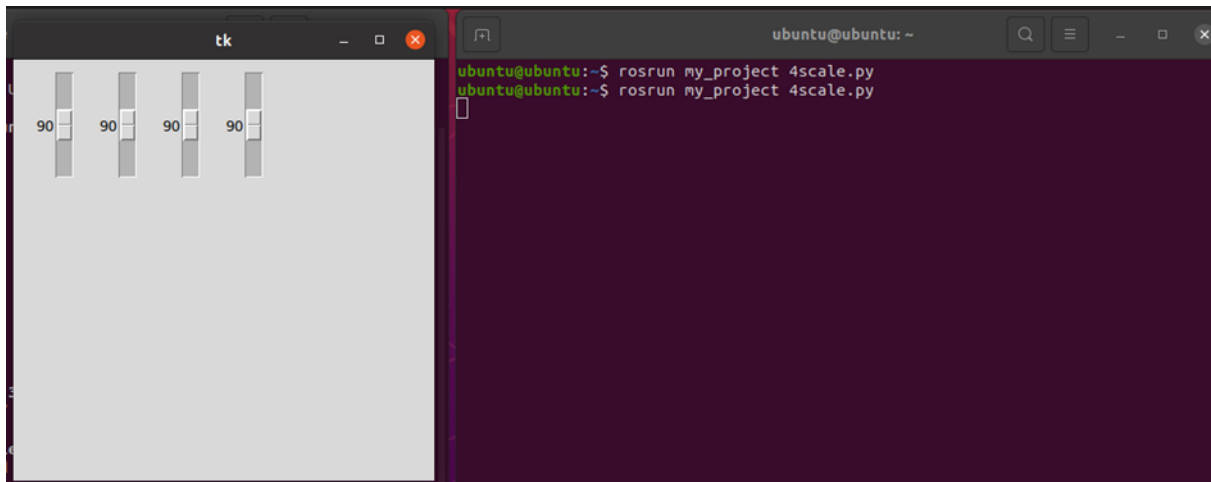
บทที่ 5

ใช้ทั้งหมดที่เรียนกับ ROBOT ARM

ทำการสร้างไฟล์ขึ้นมาชื่อ 4scale.py ในโฟลเดอร์ script และวางโค้ดลงไป

```
1  #!/usr/bin/env python3
2  from tkinter import*
3  import rospy
4  from std_msgs.msg import Int16
5
6  pub1 = rospy.Publisher('servo_1', Int16, queue_size=10)
7  pub2 = rospy.Publisher('servo_2', Int16, queue_size=10)
8  pub3 = rospy.Publisher('servo_3', Int16, queue_size=10)
9  pub4 = rospy.Publisher('servo_4', Int16, queue_size=10)
10 rospy.init_node('GUI', anonymous=True)
11 rate = rospy.Rate(100)
12
13 frame=Tk()
14 frame.geometry("400x400")
15
16 S1_value = 0
17 S2_value = 0
18 S3_value = 0
19 S4_value = 0
20
21 def talker(val):
22     global S1_value
23     global S2_value
24     global S3_value
25     global S4_value
26
27     S1_value = Int16(S1.get())
28     S2_value = Int16(S2.get())
29     S3_value = Int16(S3.get())
30     S4_value = Int16(S4.get())
31
32     pub1.publish(S1_value)
33     pub2.publish(S2_value)
34     pub3.publish(S3_value)
35     pub4.publish(S4_value)
36     rate.sleep()
37
38 S1= Scale(frame,from_=0,to=180,command=talker)
39 S1.place(x=10, y=10)
40 S1.set(90)
41
42 S2= Scale(frame,from_=0,to=180,command=talker)
43 S2.place(x=70, y=10)
44 S2.set(90)
45
46 S3= Scale(frame,from_=0,to=180,command=talker)
47 S3.place(x=130, y=10)
48 S3.set(90)
49
50 S4 = Scale(frame,from_=0,to=180,command=talker)
51 S4.place(x=190, y=10)
52 S4.set(90)
53 frame.mainloop()
```

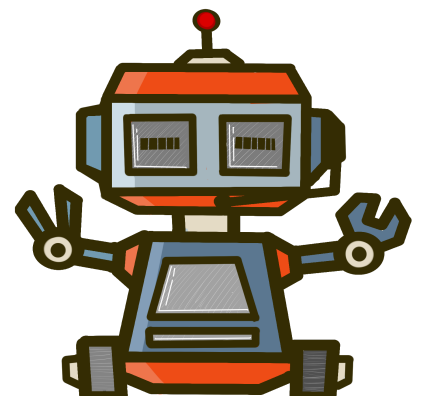
เมื่อกด run จะได้นหน้าต่างแบบนี้ ***ต้อง run roscore ในอีก terminal ก่อน***

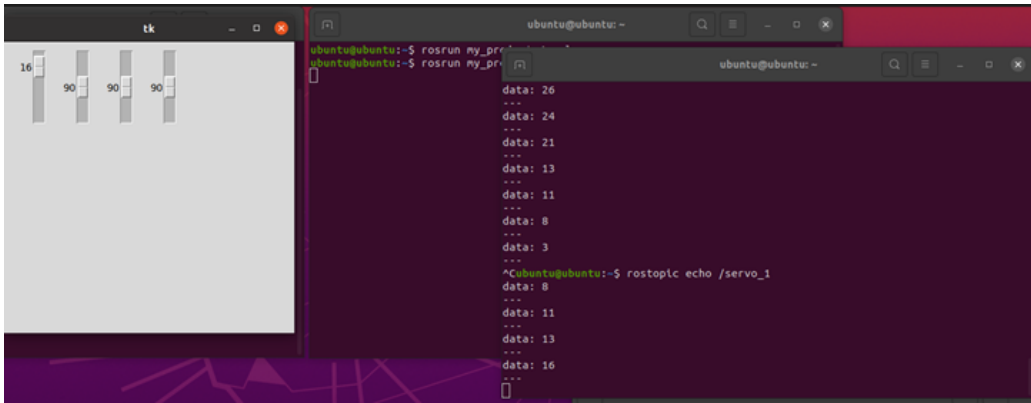


ใส่คำสั่ง rostopic list เพื่อจะดูว่ามี topic อะไรบ้าง

```
ubuntu@ubuntu:~$ rqt_graph
ubuntu@ubuntu:~$ rostopic list
/rosout
/rosout_agg
/servo_1
/servo_2
/servo_3
/servo_4
```

จากนั้นเราจะดูค่าของ topic servo_1 โดยใช้คำสั่ง rostopic echo servo_1 เมื่อ
เลื่อน scale จะพบว่าค่าจะขึ้นตาม scale ที่เลื่อน





ทำการเปิดโปรแกรม Arduino IDE ขึ้นมาและสร้างไฟล์ ชื่อ 4motor ในโฟลเดอร์และวางโค้ดนี้ลงไป

```
1 #include <Servo.h>
2 #include <ros.h>
3 #include <std_msgs/Int16.h>
4
5
6 Servo servo_1;
7 Servo servo_2;
8 Servo servo_3;
9 Servo servo_4;
10
11 void servo_cb_1( const std_msgs::Int16& cmd_msg){
12     servo_1.write(cmd_msg.data); //set servo angle, should be from 0-180
13 }
14
15 void servo_cb_2( const std_msgs::Int16& cmd_msg){
16     servo_2.write(cmd_msg.data); //set servo angle, should be from 0-180
17 }
18
19 void servo_cb_3( const std_msgs::Int16& cmd_msg){
20     servo_3.write(cmd_msg.data); //set servo angle, should be from 0-180
21 }
22
23 void servo_cb_4( const std_msgs::Int16& cmd_msg){
24     servo_4.write(cmd_msg.data); //set servo angle, should be from 0-180
25 }
26 ros::NodeHandle nh;
27
28 ros::Subscriber<std_msgs::Int16> sub_1("servo_1", servo_cb_1);
29 ros::Subscriber<std_msgs::Int16> sub_2("servo_2", servo_cb_2);
30 ros::Subscriber<std_msgs::Int16> sub_3("servo_3", servo_cb_3);
31 ros::Subscriber<std_msgs::Int16> sub_4("servo_4", servo_cb_4);
32
33
34 void setup(){
35
36     nh.initNode();
37     nh.subscribe(sub_1);
38     nh.subscribe(sub_2);
39     nh.subscribe(sub_3);
40     nh.subscribe(sub_4);
41
42     servo_1.attach(9);
43     servo_2.attach(10);
44     servo_3.attach(11);
45     servo_4.attach(12);
46 }
47
48 void loop(){
49     nh.spinOnce();
50     delay(1);
51 }
52
```

จากนั้นเราจะรันตัวส่ง ใน terminal เพื่อส่งค่าเชื่อมกับ บอร์ดนั้นไปกับตัวของ Arduino

```
ubuntu@ubuntu: ~  
During handling of the above exception, another exception occurred:  
  
Traceback (most recent call last):  
  File "/opt/ros/noetic/lib/python3/dist-packages/rospy/core.py", line 572, in s  
ignal_shutdown  
    h()  
  File "/opt/ros/noetic/lib/python3/dist-packages/roserial_python/SerialClient.  
py", line 354, in shutdown  
    self.txStopRequest()  
  File "/opt/ros/noetic/lib/python3/dist-packages/roserial_python/SerialClient.  
py", line 422, in txStopRequest  
    self.port.flushInput()  
AttributeError: 'SerialClient' object has no attribute 'port'  
ubuntu@ubuntu:~$ rosrn rosserial_python serial_node.py /dev/ttyUSB0  
[INFO] [1653620459.936576]: ROS Serial Python Node  
[INFO] [1653620459.941939]: Connecting to /dev/ttyUSB0 at 57600 baud  
[INFO] [1653620462.783041]: Requesting topics...  
[INFO] [1653620462.876341]: Note: subscribe buffer size is 280 bytes  
[INFO] [1653620462.878416]: Setup subscriber on servo_1 [std_msgs/Int16]  
[INFO] [1653620462.892187]: Setup subscriber on servo_2 [std_msgs/Int16]  
[INFO] [1653620462.903106]: Setup subscriber on servo_3 [std_msgs/Int16]  
[INFO] [1653620462.918329]: Setup subscriber on servo_4 [std_msgs/Int16]  
S
```

เราได้ทำการทดสอบโปรแกรมในกาควบคุมข้อต่อของแขนหุ่นเป็นอันเสร็จสิ้น.

