uArm Swift Pro robotic arm motion and action manual

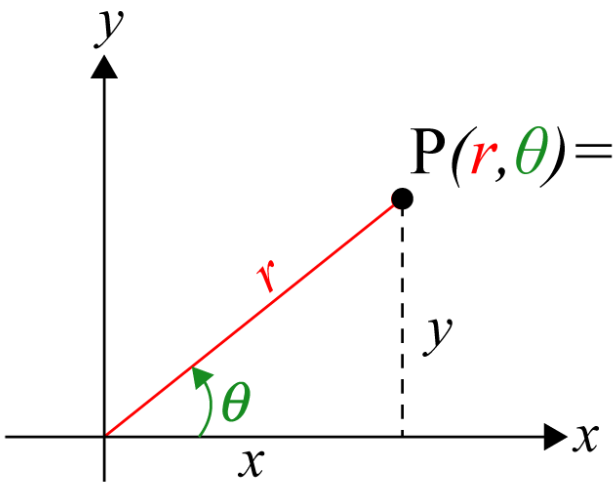
Created by

Panitan Kwankaew

School of Engineering Bangkok University

General understanding of the Polar Coordinate system

Two-dimensional polar coordinate system



$$x = r\cos\theta$$
$$y = r\sin\theta$$
$$x^2 + y^2 = r^2$$
$$\tan\theta = \frac{y}{x}$$

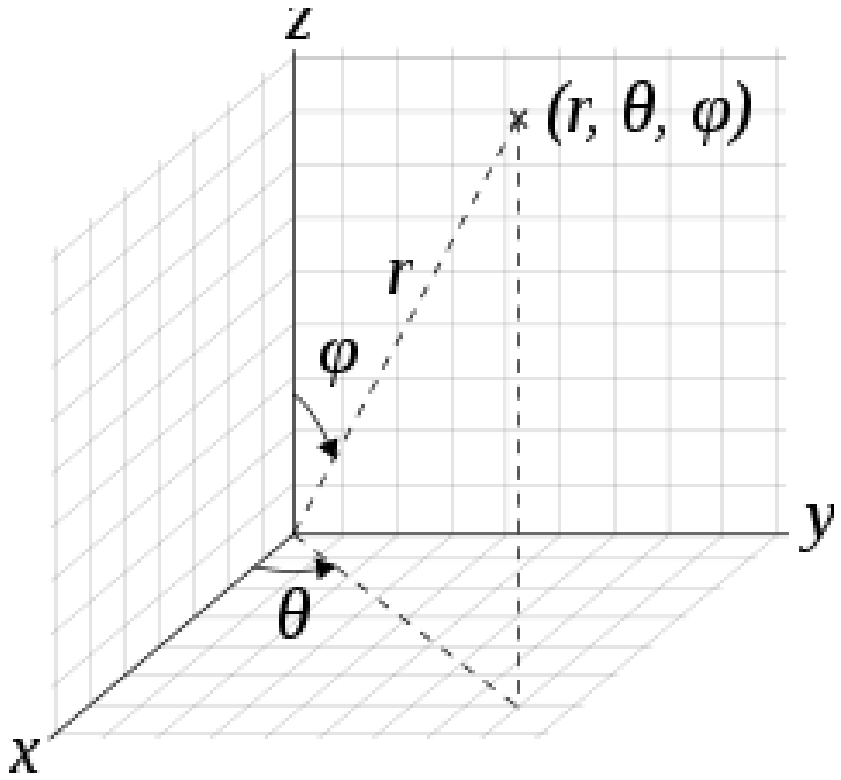
Calcworkshop.com

The x coordinate and y coordinate of a point of interest can be represented by the distance from the origin and an angle made with the x-axis here represented by r and Θ (theta) accordingly.

Using trigonometry we can summarize the relation between polar coordinate and rectangular coordinate as

P(x, y) = P (r cosΘ, r sinΘ) = P (r, Θ)

Three-dimensional polar coordinate system



The three-dimensional polar coordinate system can also be called a spherical coordinate system or Cartesian coordinate.

The x, y, and z coordinates of a point of interest can be represented by the distance from the origin and an angle made with the x-axis and z-axis
here represented by r, Θ (theta), and $\boldsymbol{\Phi}$ (phi) accordingly.

Using trigonometry we can summarize the relation between 3D polar coordinate and rectangular coordinate as

$$P(x, y, z) = P(r\cos\boldsymbol{\Phi}\sin\Theta,\ r\sin\boldsymbol{\Phi}\sin\Theta,\ r\cos\Theta) = P(r, \Theta, \boldsymbol{\Phi})$$

uArm Swift Pro coordinate systems

In uArm Swift Pro, there are 2 available coordinate systems to use

Rectangular coordinate system

Using the tip of an arm as a point of interest

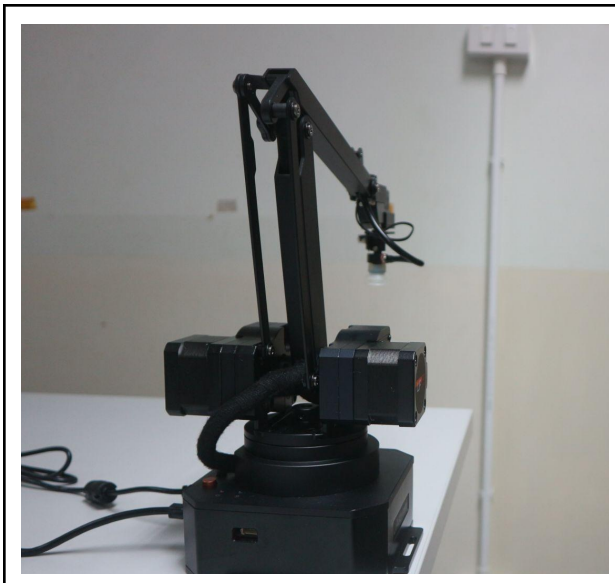
At its initial position, the rectangular coordinate is

P (200, 0, 150)

```
send in direction
 Coordinate in polar = :
[199.91, 90.0, 150.03]


 Coordinate in axial = :
[199.91, -0.0, 150.03]
```

At its rightmost position, the rectangular coordinate is
P (0, -190, 150)

```
5
send in direction
 Coordinate in polar = :
[192.05, 0.0, 149.89]


 Coordinate in axial = :
[0.0, -192.05, 149.89]
```

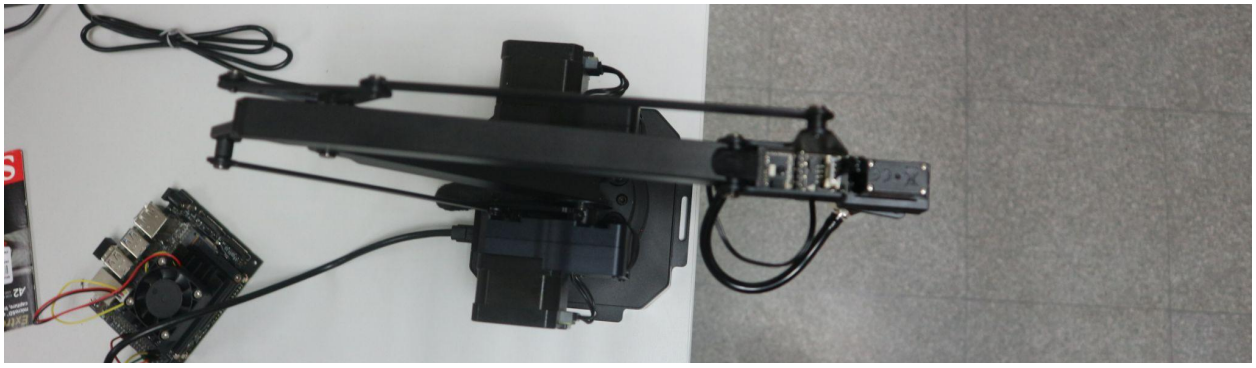


At its leftmost position, the rectangular coordinate is
P (0, 190, 150)

```
send in direction
 Coordinate in polar = :
[195.35, 180.0, 149.99]


 Coordinate in axial = :
[0.0, 195.35, 149.99]
```

With maximum arm stretch in perpendicular with y-axis,
the rectangular coordinate is P (350, 0 , 100)

```
send in direction
 Coordinate in polar = :
[350.76, 90.0, 99.35]


 Coordinate in axial = :
[350.76, -0.0, 99.35]
```



With minimum arm stretch in perpendicular with y-axis,
the rectangular coordinate is P (140, 0 , 130)

```
send in direction
 Coordinate in polar = :
[138.84, 90.0, 129.58]


 Coordinate in axial = :
[138.84, -0.0, 129.58]
```

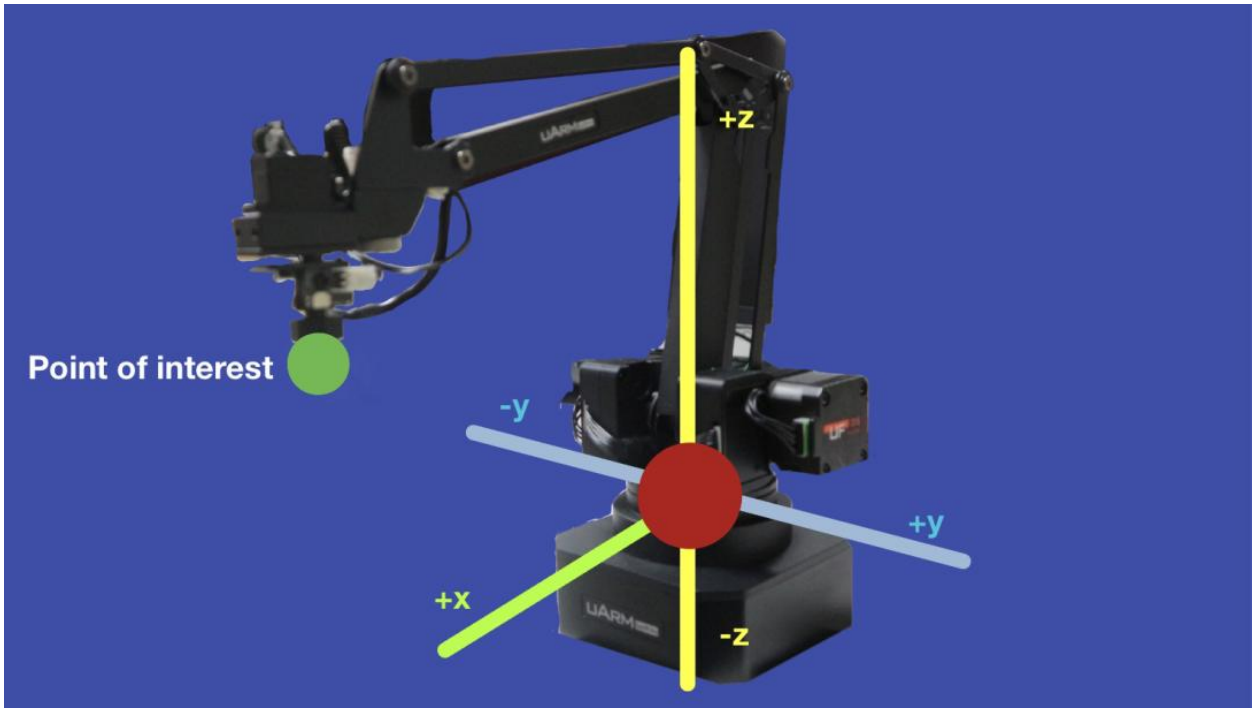
At the maximum distance from the ground,
the rectangular coordinate is P (194, 0, 170)

```
 Coordinate in polar = :
[194.98, 90.0, 170.07]


 Coordinate in axial = :
[194.98, -0.0, 170.07]
```

At the minimum distance from the ground,
the rectangular coordinate is P (215, 0, -125)

```
send in direction
 Coordinate in polar = :
[215.0, 90.0, -124.76]


 Coordinate in axial = :
[215.0, -0.0, -124.76]
```

The conclusion is that the x-axis, y-axis, and z-axis can be drafted according to this diagram



Red ball is an origin (0, 0, 0)

According to the experiment we can estimate the boundary of each axis

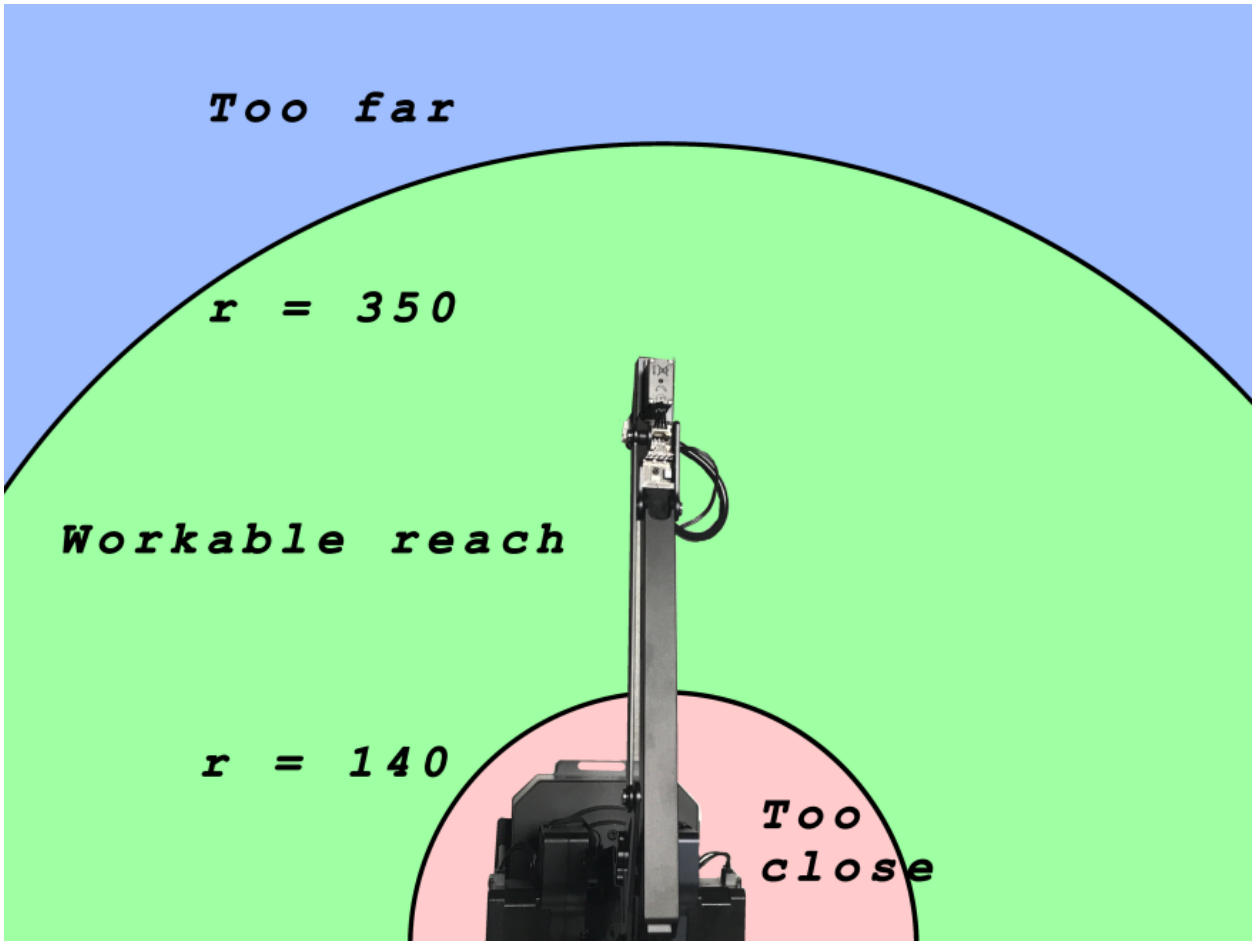
for x ∈ [0, 350], y ∈ [-190, 190], and z ∈ [-120, 170]

However this is not the most practical coordinate system to use in uArm due to its limitation in tracking, programming, and real-life application since the uArm will need exact coordination to make a move. This rectangular coordinate system works just fine with constrained works or in a square planar work environment but faces difficulty in tracking or any dynamic works.

The 2D polar coordinate system with z-axis

Using the tip of an arm as a point of interest

In this mode the uArm will see itself as a steady body in the center of a semi-circle and its tip will be a point of interest.



The parameter of coordinates will be [r, Θ, z] Θ will increase in the anti-clockwise direction its initial position is [200, 90, 150], z coordinate is identical to the one used in a rectangular one,

r = $\sqrt{x^2 + y^2 + z^2}$, the theta is an angle made on XY plane. There is a function that can get both polar and axial coordinates so there is no need to be concerned with it.

Programming

Motion programming

In uArm Swift Pro there is a few choices of programming language to choose from including C++ and Python which come in a library for you to download before using them. In this manual Python is the selected one. You can get the Python library from the following link: https://github.com/uArm-Developer/uArm-Python-SDK.

There is an executable setup.py file simply located and execute to install.

There 2 available coordinate systems for you to use in uArm but, since a rectangular one is not practical in this manual only 2D polar coordinates with z will be used.

A keyboard will be used as an input in this example which can be downloaded

from this link: https://github.com/PanitanK/BIGG.git.

Code

Header section

These following libraries can be found in the original contributor any further details can probably be found in his git.
I simply copied his imported libraries and it worked just fine.
The coordination parameter [r, θ, z] are represented as stret(ch), rotate, and high variables accordingly.
[stret , rotate, high] will be the parameter we will use.

Movement

```
swift.waiting_ready(timeout=20)
device_info = swift.get_device_info()
firmware_version = device_info['firmware_version']
if firmware_version and not firmware_version.startswith(('0.', '1.', '2.', '3.')):
    swift.set_speed_factor(0.0005)
swift.set_mode(0)
swift.set_polar()
```

swift.waiting_ready at the beginning of every action was used to set a timeout in order to prevent disconnection from delay made by either software or hardware.

```
device_info = swift.get_device_info()
firmware_version = device_info['firmware_version']
if firmware_version and not firmware_version.starts with(('0.', '1.', '2.', '3.')):
```

These 3 lines were used to get the status of a uArm if it is available for action and firmware_version was there to check if the firmware is obsolete or not most of the time there won't be many issues with these since uArm firmware lasted update was 4 years ago so it is stable as it is.

However, these can't be skipped I've tried and it causes an error you can try to shorten it as a function but I didn't so it's your choice if you want to simplify the code.

swift.set_mode(0) is a function that make a motion possible (enable rectangular / polar coordinates movement)

swift.set_ploar is a function that move according to the parameter [stret , rotate, high] if the parameter is empty the default value [200 , 90 , 150] will be set.

```
elif a == "-" :
    swift.waiting_ready(timeout=20)
    device_info = swift.get_device_info()
    firmware_version = device_info['firmware_version']
    if firmware_version and not firmware_version.startswith(('0.', '1.', '2.', '3.')):
        swift.set_speed_factor(0.0005)
    print(a)
    print("send in direction")
    swift.set_mode(0)
    stret = stret - 5
    swift.set_polar(rotation = rotate , height= high , stretch=stret)

    print("direction online")
```

By editing the value inside the parameter I can alter the position of the uArm tip.

Detach/Attach the servos

```
    elif a == "d" :
        i = i % 2
        if i == 1 :
            print(swift.set_servo_detach())

        else :
            print(swift.set_servo_attach())
        i = i+1
        time.sleep(2)
```

Detaching and Attaching the servo can be done in a single function unlike other movement functions this because attaching and detaching is a simple task by putting servos into a neutral.

Be advised detaching can cause damage if the arm is in an unstable position

Get position

```
elif a == "ctrl" :
    print("set")
    swift.waiting_ready(timeout=20)
    device_info = swift.get_device_info()
    firmware_version = device_info['firmware_version']
    if firmware_version and not firmware_version.startswith(('0.', '1.', '2.', '3.')):
        swift.set_speed_factor(0.0005)

    print("send in direction")
    swift.set_mode(0)

    pocor = swift.get_polar()

    print("\n")
    print(" Coordinate in polar = : " )
    print(pocor)
    stret = pocor[0]
    rotate = pocor[1]
    high = pocor[2]

    swift.set_polar(rotation = rotate , height = high , stretch = stret)
    print("\n")
    print("direction online")
```

swift.get_polar and swift.get_position will send the callback as an array of [r, θ, z] for get_polar and [x, y, z] for get_position.

The value inside an array can be separate and assigned to each this is necessary for any streamlined process to update the position when you reattach the uArm in a different position else the arm will navigate to the position in its memory instead of continuing the current position.