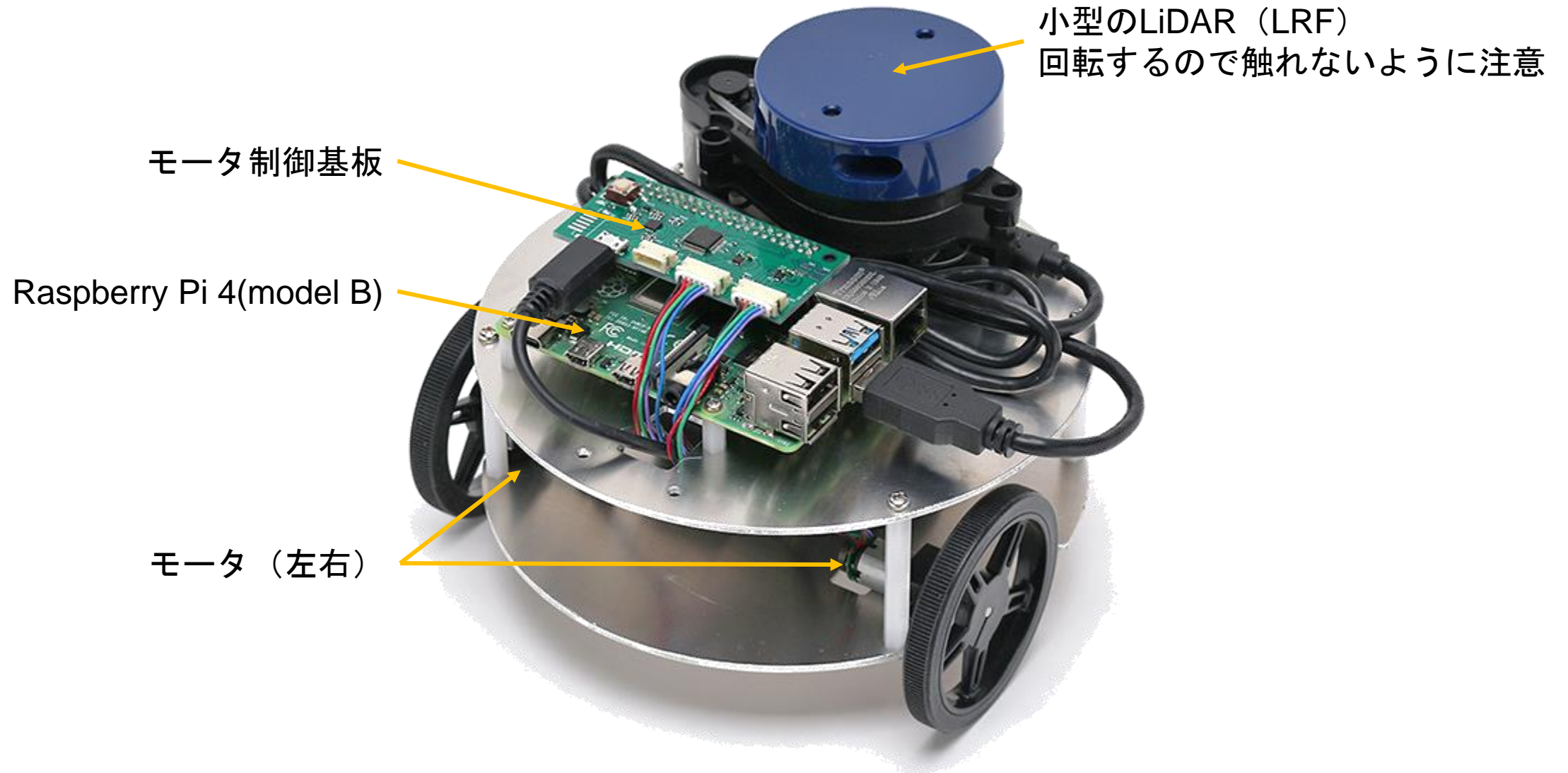


知能ロボット実習

第1回

ロボットとROS (Robot Operating System)

ロボット: ライトローバー (VSTON)



実習目的

- Linux, ROS (Robot Operating System), Python, コンピュータ連携
- センサ情報処理, 確率的状態推定, 画像による認識, ニューラルネットワークによる制御

実習内容

- ロボットとROS (Robot Operating System) (第1回)
 - オドメトリ情報, LiDAR情報の取得
 - オドメトリ情報による自己位置推定
 - LiDAR情報による自己位置推定
 - 性能比較, 結果解析
- 地図作成と自己位置推定 (第2回)
 - 地図作成 (遠隔操作)
 - 地図によるナビゲーション (経路をコマンド指定)
 - 確率的状態推定
 - 自律ナビゲーション

実習内容

- センサの複合処理による自律動作（第3回）
 - 画像による認識
 - 画像認識による自律動作
 - ニューラルネットによる制御
- 自由課題（第4回）

実習目的

- ロボットのプログラミング技術を習得する
- 各種センサの情報処理方法を習得する
- チームでロボットの情報処理システムを構築できるようにする
- 課題の設定から解決方法の実装までをできるようにする

ROS (Robot Operating System)

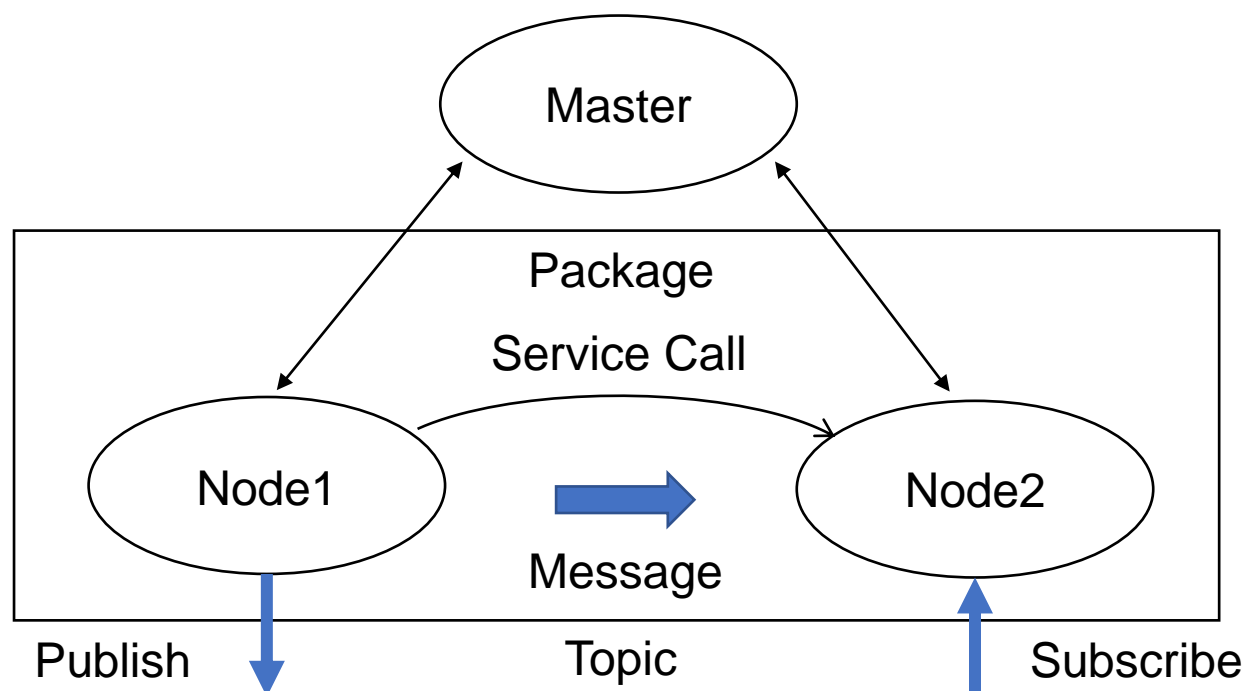
- ロボット開発に用いるオープンソースソフトウェア
- スタンフォード大学の学生が開発した「Switchyard」プロジェクトが起源
- それを引き継いだアメリカの「Willow Garage」が2007年に本格開発を開始
- 2010年1月に最初のリリース版「ROS 1.0」がリリース
- その後, 「OSRF」(Open Source Robotics Foundation)が設立, ROSの開発を主導

ROSの特徴



- Plumbing: 出版購読型（Publish/Subscribe）の通信モデルとミドルウェア
- Tools: プロジェクト管理, デバッグ, 可視化, など
- Capabilities: 充実したライブラリやパッケージ
- Ecosystem: 世界規模のOSS（Open Source Software）コミュニティ

ROSの通信方式



- 複数のプログラムを結合させる「通信ライブラリ」がROSの中心
- ROSの実行プログラムはNode（ノード）という単位で扱う
- Publisher / Subscriber型で通信
 - 通信相手が誰かを意識することなく通信可能
 - Topicは型が決まっているため、相手のIPアドレスやポート番号などの情報なしに通信可能
- Serviceという同期的な通信（「呼び出し」が終わるまで「ブロック」する）もある

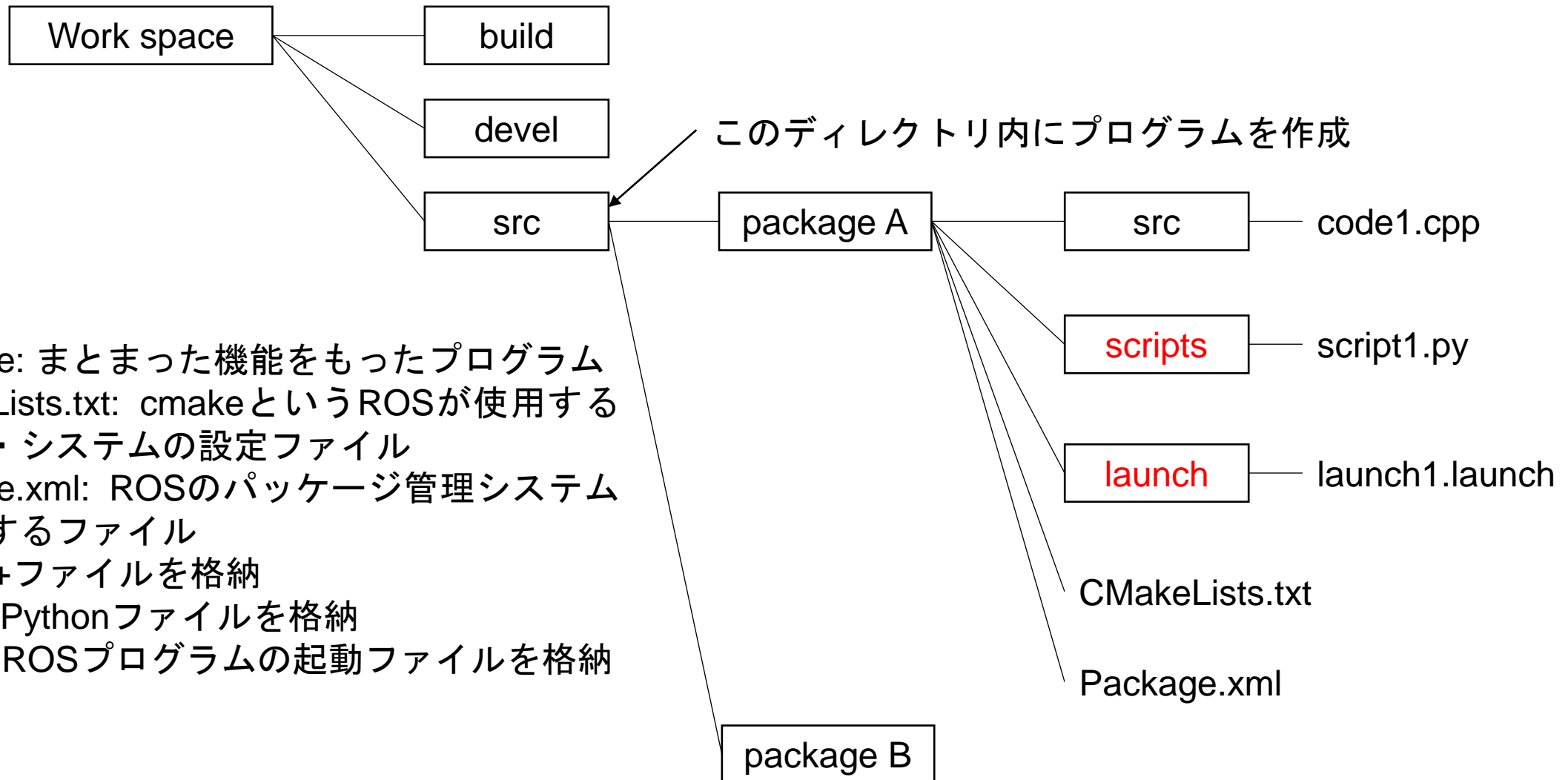
ROSの用語

マスター(master)	ノードやトピックの管理をするプロセス
ノード(node)	ロボット制御システムを構成する一つのプロセス
パッケージ(package)	特定の機能を実現するためのノードの集まり
トピック(topic)	publish/subscribe型の非同期のノード間通信方法
メッセージ(message)	トピックでやり取りされるデータ
パブリッシャー(publisher)	トピックを送信（publish）するノード
サブスクライバー(subscriber)	トピックを受信（subscribe）するノード
サービス(service)	リクエスト/リプライ型のノード間通信方法
パラメータ(parameter)	ノード外から動的に変更可能なノード内変数

ROSの使用言語

- メインはPythonまたはC++
- ROSは複数のノードを協調動作させて一つのシステムを構成
- それぞれのノードを異なるプログラミング言語で作成可能
- 処理速度を重視したい場合はC++を使用
- Pythonのライブラリを使用したい場合はPythonを使用
- 今回はPythonを用いる

ROSシステムのディレクトリ構成



Python

- 記述方式
 - インデントの利用
 - オブジェクト指向
- 幅広い用途
 - 機械学習
 - データ収集・解析
 - アプリケーション開発
- 豊富なライブラリ
 - Numpy : 数値計算ライブラリ
 - Pandas : データ分析ライブラリ

Pythonサンプルコード

print 標準出力に表示

```
# coding: utf-8
print("Test")
print(100)
print([0, 1, 2])
print( 'Time is %d:%d' % (17, 30))
```

文字列の操作

```
# coding: utf-8
a="Hello"
b="world"
c=a+b
print("%s+%s=%s" % ( a, b, c ) )
```

Pythonサンプルコード

配列 (list)

```
# coding: utf-8
var_list=[1, 2, 3, 4, 5]
print(var_list)
var_list2=[' a', ' abd', 3, 0.1, ' a' ]
print(var_list2)
print(var_list2[1])
print(var_list2[-1])
```

繰り返し処理 (for文)

```
# coding: utf-8
vlist=[' a', ' b', ' c' ]

for i in vlist:
    print( i ) ← インデントに注意

for i in range(0, 10):
    print( i )
else:
    print("loop end")

for i, val in enumerate(vlist):
    print( i, val)
```

Pythonサンプルコード

条件分岐 (if文)

```
# coding: utf-8
a=1
if( a==1 ):
    print("a=1" ) ← インデントに注意

b=1
if( a==1 and b==1 ):
    print( "a=1 and b=1" )
elif( b==0 ):
    print( "b=0" )
else:
    print( "other cases" )
```

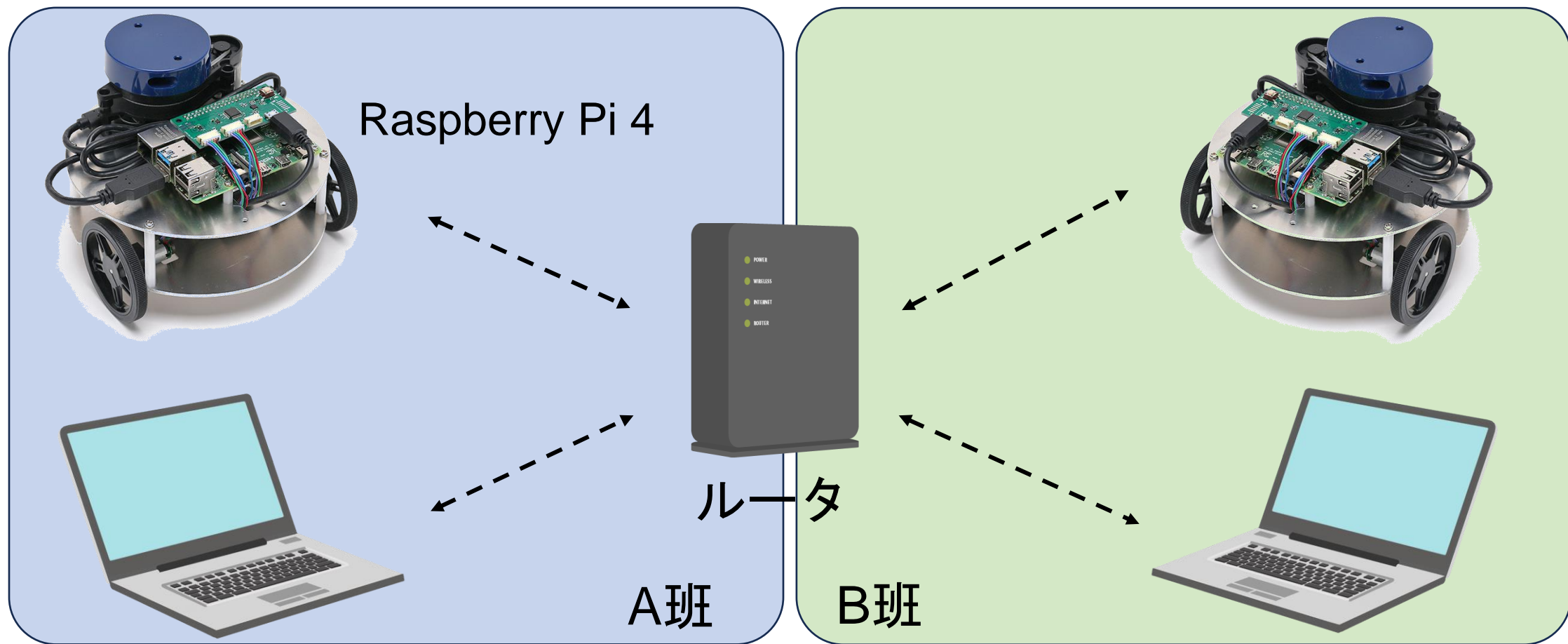
関数

```
# coding: utf-8

def test1():
    print("test")
test1()

def test2(a,b):
    return a + b
num = test2(2,3)
print(num)
```


ロボットのリモート開発



- ロボットのコンピュータ（Raspberry Pi 4）にリモート接続
- プログラム開発


リモート接続の設定

- wi-fi (Buffalo-G-C-140) に接続する
- IPアドレスを固定する
 - Windowsの場合 → スライド17ページ
 - Mac OSの場合 → スライド21ページ
- 授業終了後は、IPアドレスの固定を解除してください
 - 解除しない場合、インターネットに接続できなくなるので注意



すべて アプリ ドキュメント ウェブ その他 ▼

最も一致する検索結果

 ネットワーク接続の表示
コントロール パネル

Web の検索

- 🔍 ネットワーク接続の表示 - Web 結果を見る >
- 🔍 ネットワーク接続の表示 windows10 >
- 🔍 ネットワーク接続の表示が出ない >
- 🔍 ネットワーク接続の表示 コマンド >
- 🔍 ネットワーク接続の表示 ショートカット >
- 🔍 ネットワーク接続の表示 10 >
- 🔍 ネットワーク接続の表示確認 >
- 🔍 ネットワーク接続の表示 win10 >

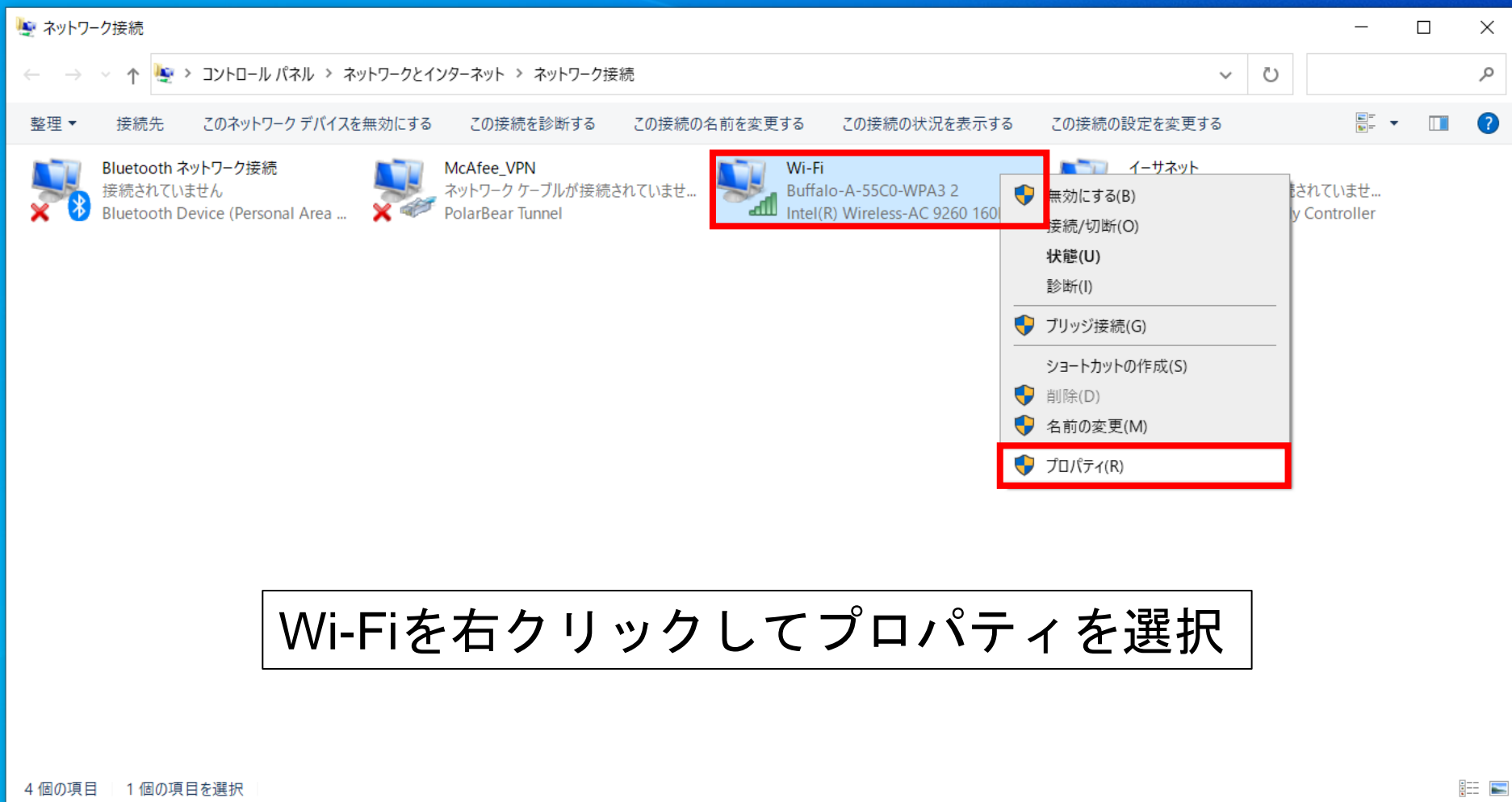
ネットワーク接続の表示
コントロール パネル

🔗 開く

スタートメニューから「ネットワーク接続の表示」
を検索して選択する
Windows11の場合も同様

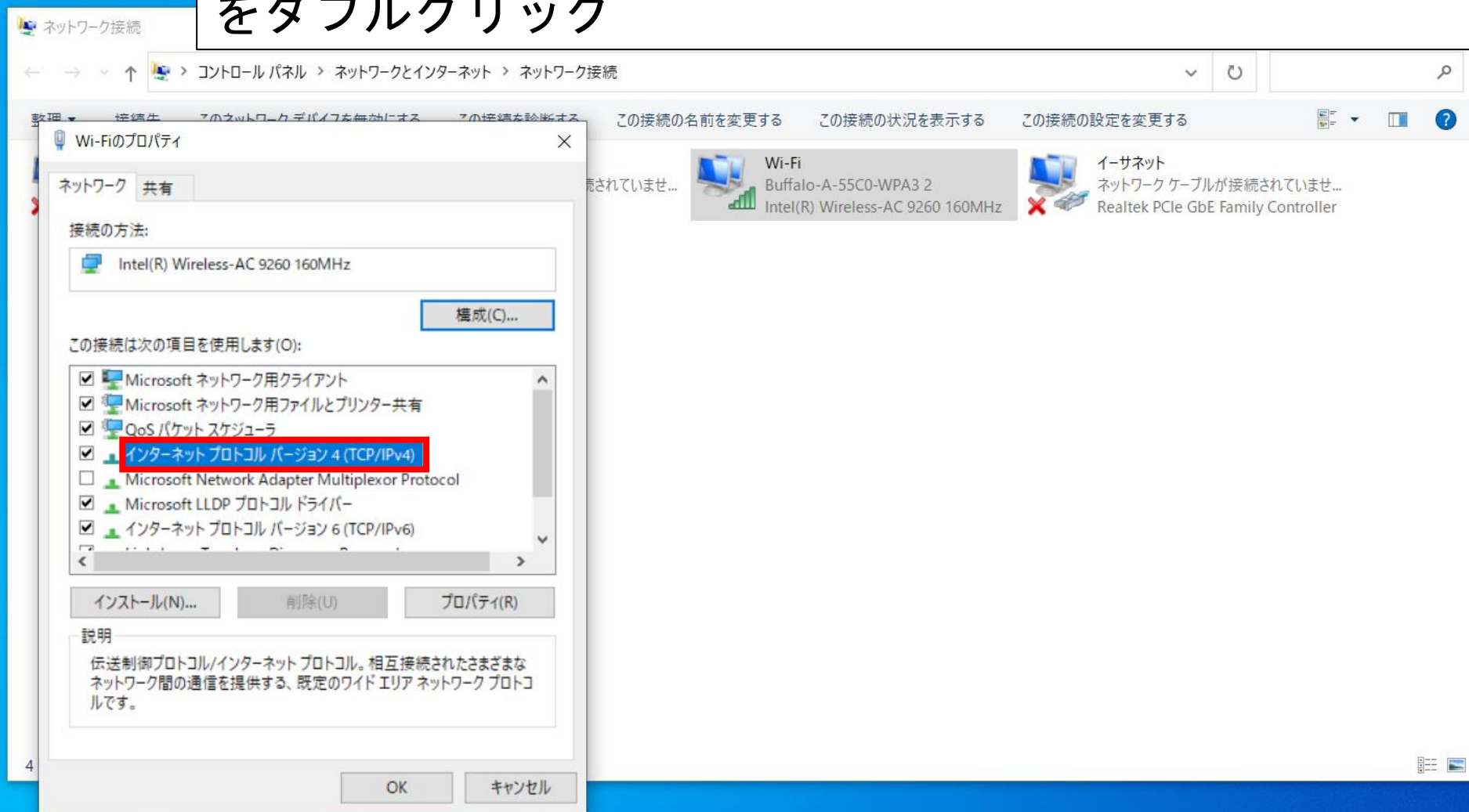
🔍 ネットワーク接続の表示





Wi-Fiを右クリックしてプロパティを選択

「インターネット プロトコル バージョン 4 (TCP/IPv4)」 をダブルクリック



「次のIPアドレスを使う」を選択
IPアドレスに「192.168.70.(自分の出席番号)」を入力
(出席番号が1～7番の学生は101～107に設定)
サブネットマスクは自動入力される
他は空白でよい
OKを押したら完了

ネットワーク接続

コントロール パネル > ネットワークとインターネット > ネットワークと共有センター > Wi-Fiのプロパティ

ネットワーク 共有

インターネット プロトコル バージョン 4 (TCP/IPv4)

全般

ネットワークでこの機能がサポートされている場合は、IP 設定を自動的に取得することができます。サポートされていない場合は、ネットワーク管理者に適切な IP 設定を問い合わせてください。

☐ IP アドレスを自動的に取得する(O)

☒ 次の IP アドレスを使う(S):

IP アドレス(I): 192 . 168 . 70 . |

サブネット マスク(U): 255 . 255 . 255 . 0

デフォルト ゲートウェイ(D): . . .

☐ DNS サーバーのアドレスを自動的に取得する(B)

☒ 次の DNS サーバーのアドレスを使う(E):

優先 DNS サーバー(P): . . .

代替 DNS サーバー(A): . . .

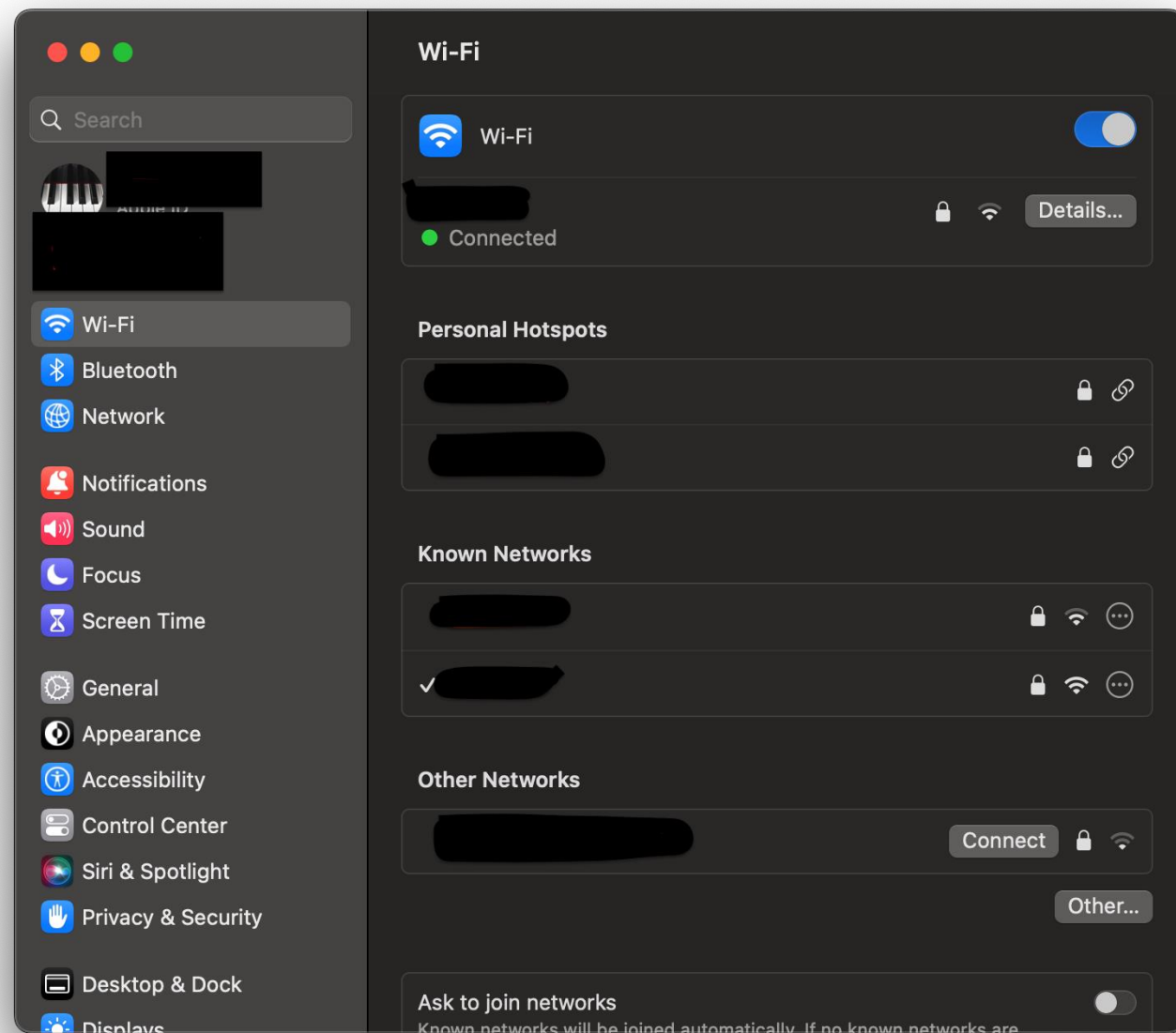
☐ 終了時に設定を検証する(L)

詳細設定(V)...

OK キャンセル

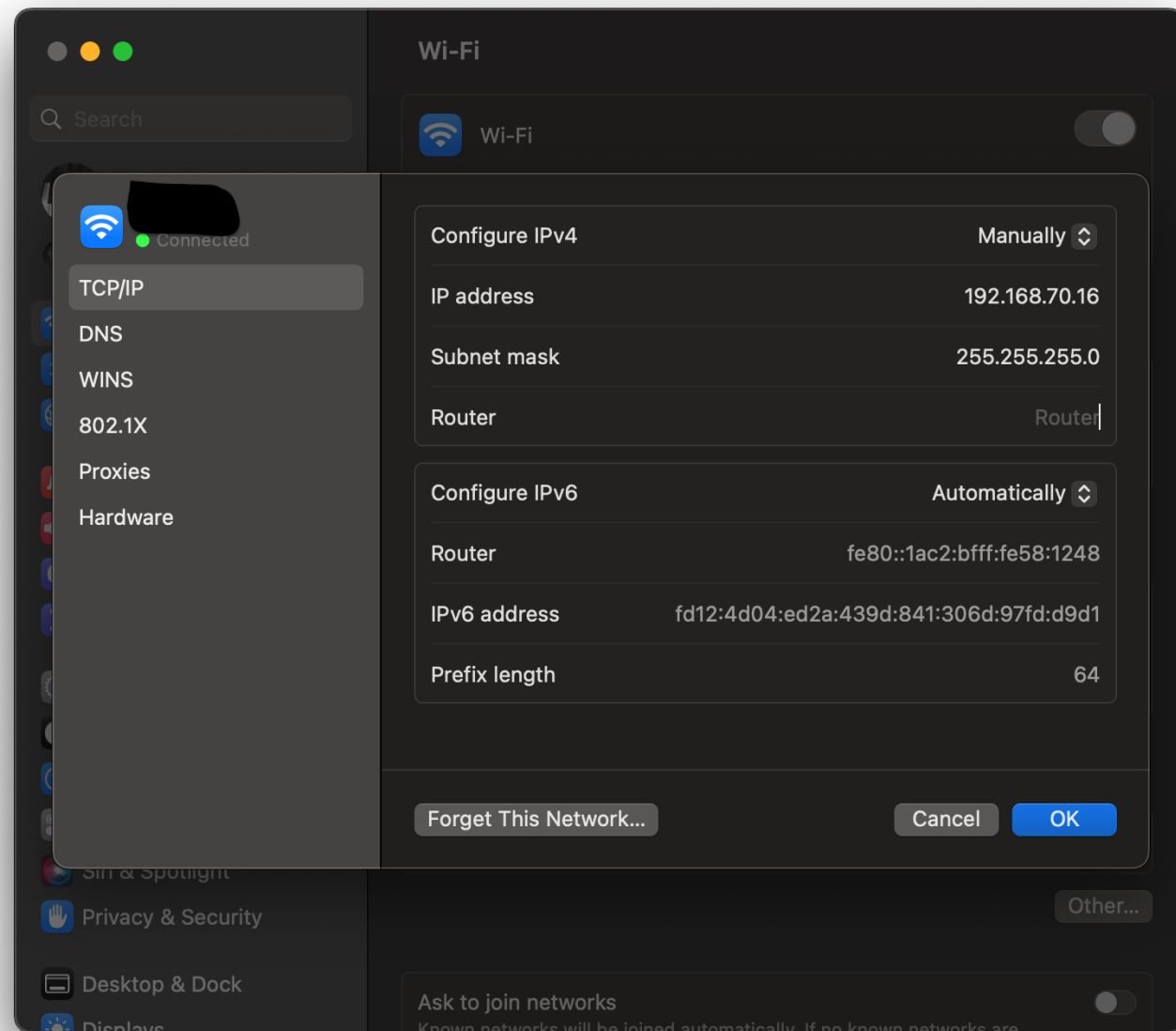
IPアドレスの固定方法（Mac OS）

- wi-fiの設定画面を開く
- wi-fiに接続
- Details（詳細）をクリック



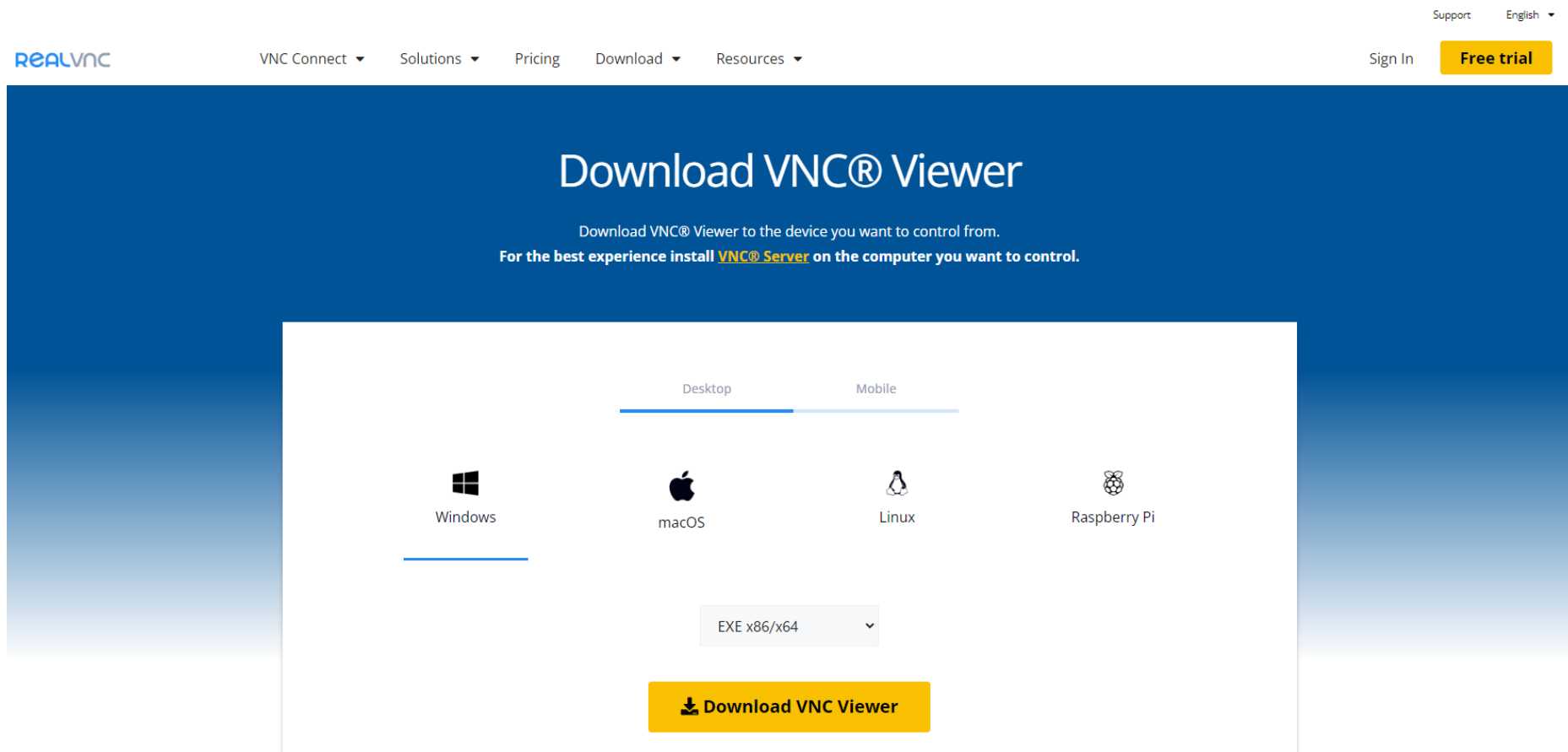
IPアドレスの固定方法（Mac OS）

- TCP/IP をクリック
- IPv4の設定をマニュアルにする
- IP addressを192.168.70.(出席番号)に設定（出席番号が1～7番の学生は101～107に設定）
- Subnet maskを255.255.255.0に設定
- OKをクリック



ロボットのリモート開発・操作

VNCを使用してリモートでロボットのソフトウェア開発・操作

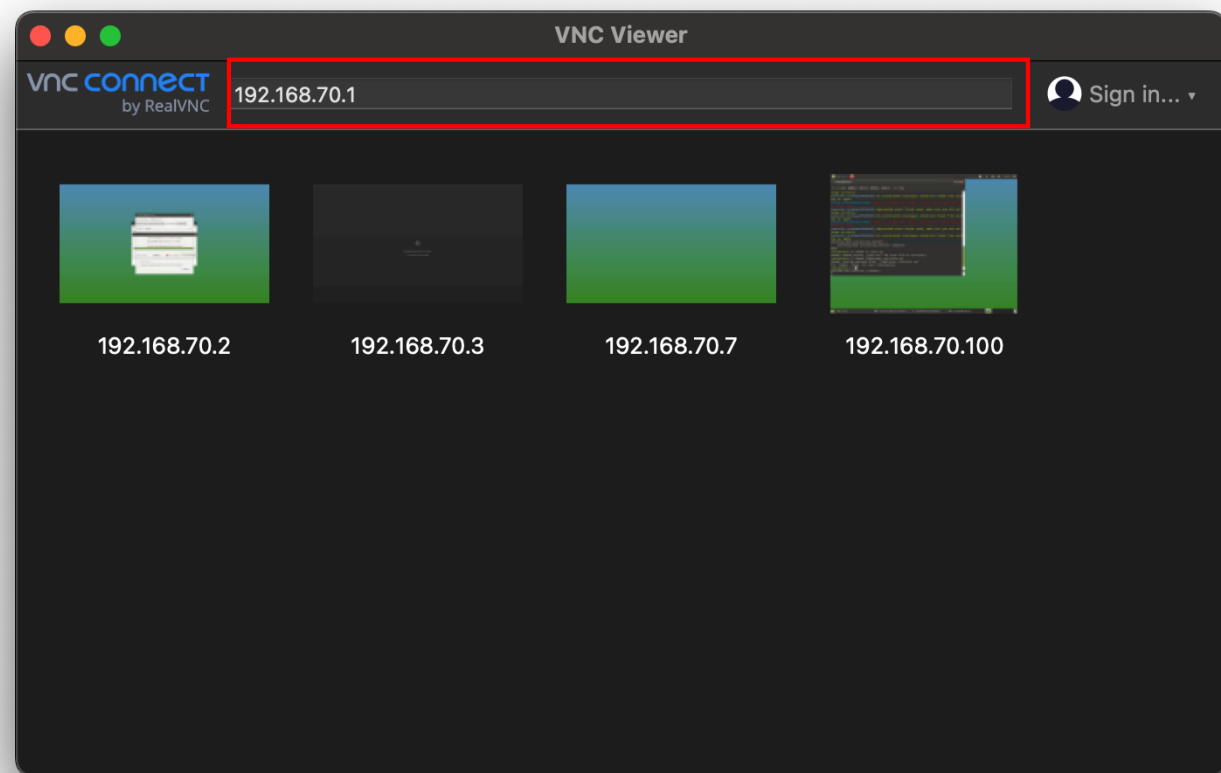


<https://www.realvnc.com/en/connect/download/viewer/>

手順に沿ってインストール

ロボットのリモート接続方法

- VNCを起動するとアカウントの作成を求められるが作成しなくて良い
- 上部のボックスに192.168.70.(ロボットの番号) を入力
- Continueをクリックする
- パスワードをrobotと入力



ROSの実行手順

- roscore起動
 - ROS masterを起動するコマンド
 - 必ず起動する必要がある
 - ターミナルを起動して, roscoreを実行
- ノード起動(rosrun)
 - ターミナルを起動して, rosrn [パッケージ名] [ノード名]
- roslaunch
 - 複数のノードを同時に起動
 - ターミナルを起動して, roslaunch [パッケージ名] [launchファイル名]

オドメトリとLiDAR

- オドメトリ

- 車輪の回転から進行速度, 回転速度を計算し, 位置の変化を推定する

- LiDAR

- 光波を照射して周囲に存在する物体の相対位置を計測する

オドメトリデータ

- Odometry.py (行数を表示)

- (137) odom.pose.pose.position.x
- (138) odom.pose.pose.position.y
- (142) odom.pose.pose.orientation.z
- (146) odom.twist.twist.linear.x
- (148) odom.twist.twist.angular.z

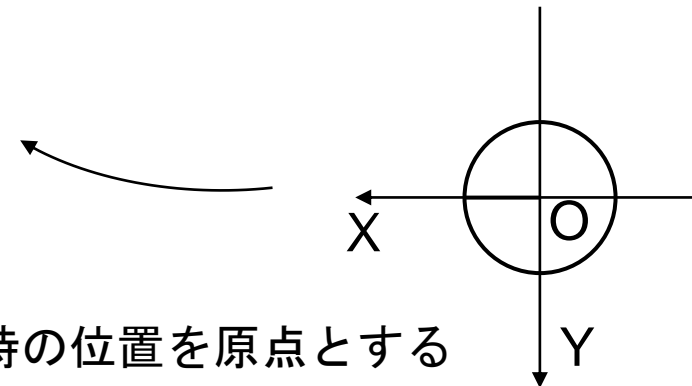
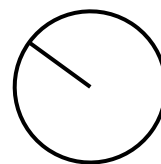
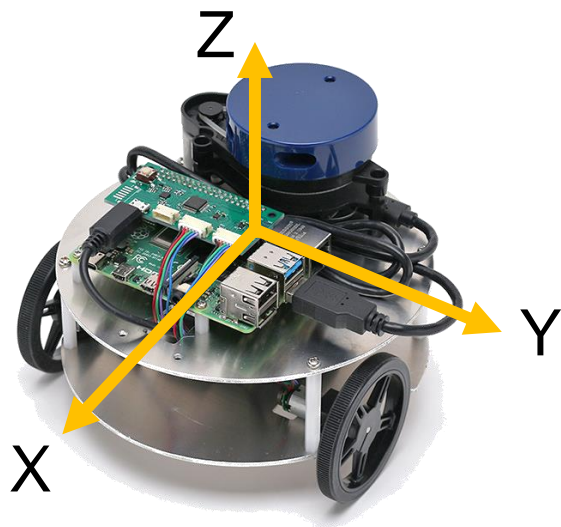
スタートからの相対位置 (X方向) [m]

スタートからの相対位置 (Y方向) [m]

スタートからの相対方位 (X方向を0[rad])

前進速度 [m/s]

回転速度 [rad/s]



プログラム実行時の位置を原点とする

オドメトリデータ取得方法

- pid_controller.py（行数を表示）
 - (66) rospy.Subscriber('odom', Odometry, get_rover_v)
odomを受信すると関数get_rover_vを呼び出す
コールバック関数と呼ぶ
 - (34) linear_x = data.twist.twist.linear.x
 - (35) angular_z = data.twist.twist.angular.z
関数get_rover_vで前進速度と回転速度を取得している

ロボットへの速度指令

- pid_controller.py (行数を表示)
 - (43) output = pid_controller.pid_controll(current_v, target_rover_v)
- target_rover_vの値を入力
0: 前進速度, 1: 回転速度

最大値の目安

前進速度は0.15 [m/s]以下

回転速度は0.349 [rad/s]以下 (20 [°/s])

if文で最大値を超えないようにする

LiDARデータ取得方法

```
#!/usr/bin/env python3
import rospy
from sensor_msgs.msg import LaserScan #レーザスキャンを扱うメッセージ

def callback_laser(msg): #コールバック関数と呼ぶ
    i = 0
    range = msg.ranges[i]
    print(range) #抽出した距離を表示

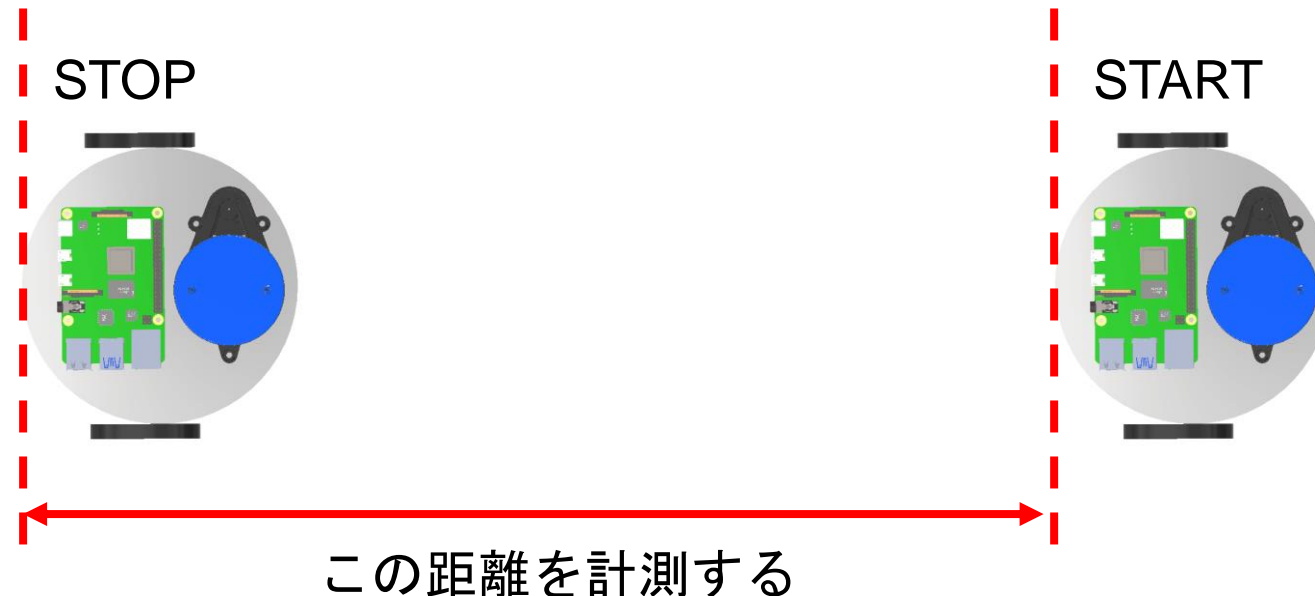
rospy.init_node('range_test') #'range_test'というノードを生成・初期化

scan_sub = rospy.Subscriber('scan', LaserScan, callback_laser)
#'scan'トピックをLaserScan型でsubscribeして, callback_laser関数を呼び出す

rospy.spin() #停止コマンドを受けるまで終了しない
```


課題 1

オドメトリによって自己位置を推定しつつ，指定の距離（0.1m）進んで停止するプログラムを記述せよ．繰り返し（10回）行うことで，停止位置のばらつき（標準偏差）を定規を用いて計測せよ．



課題 1 のヒント

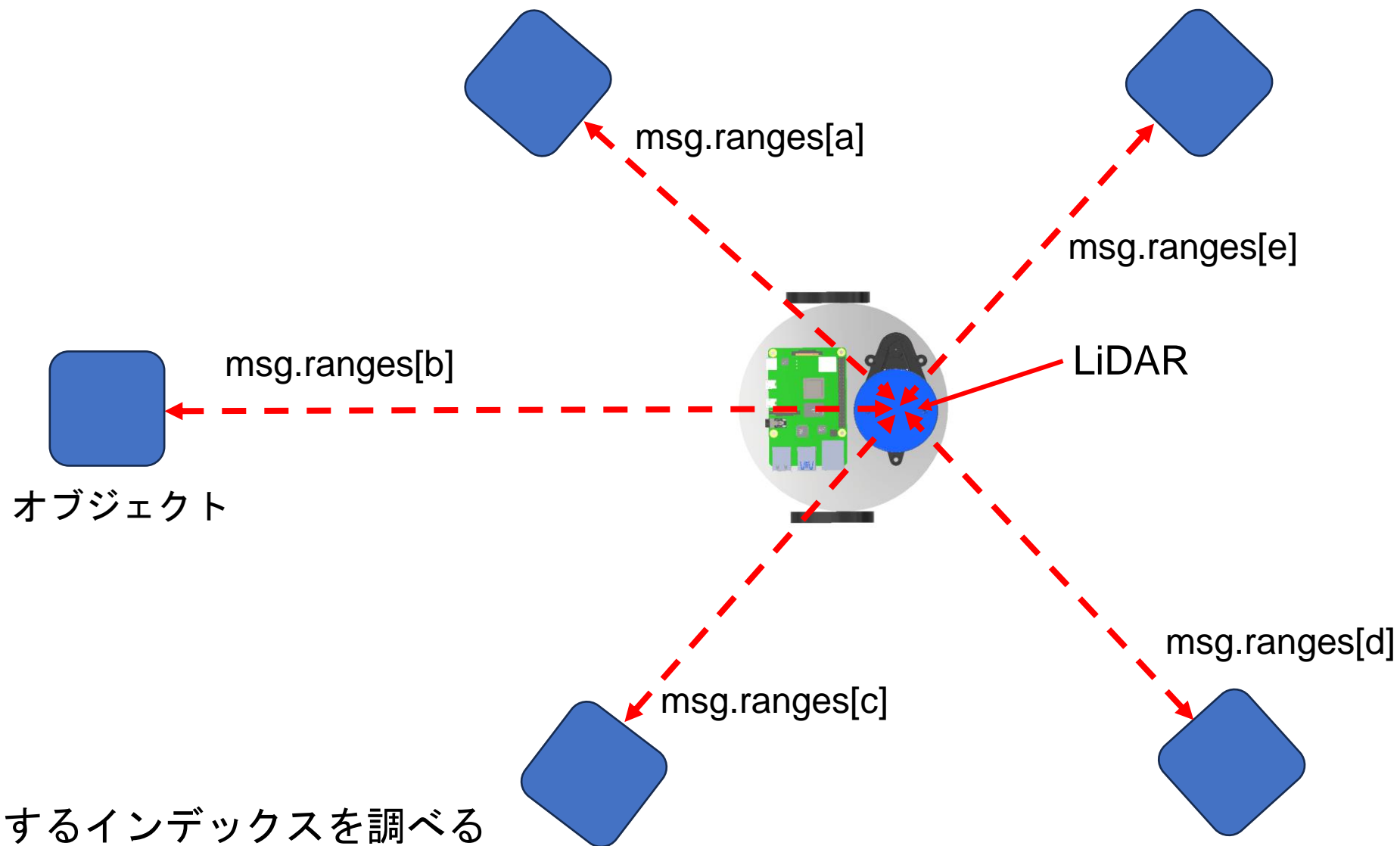
- /home/robot/catkin_ws/src/lightrover_ros/scripts/**rover_controller.py** を編集
- 関数**controller()**内を編集
 - odom_x : 進んだ距離[m] (オドメトリの値)
 - speed.linear.x : 指令前進速度[m/s]
 - speed.angular.z : 指令回転速度[rad/s] **単位に注意してください**
- Pythonでの if 文の書き方を調べましょう
- ターミナルを起動 (**Ctrl + Alt + t**)
- **roslaunch lightrover_ros controller.launch** で実行可能
- ロボットの停止は **Ctrl + C** で行う
- 課題ごとにターミナルでscriptsディレクトリに入って以下で実行権限を付与してください
 - **chmod 755 *.py**

課題 2

LiDARのデータを取得して、ロボットの前方にあるオブジェクトまでの距離を表示するプログラムを記述せよ.

`msg.ranges`の配列の各要素が周囲の距離情報に相当するが、各要素がどの方向の距離に対応するか調べよ.

課題 2 のヒント



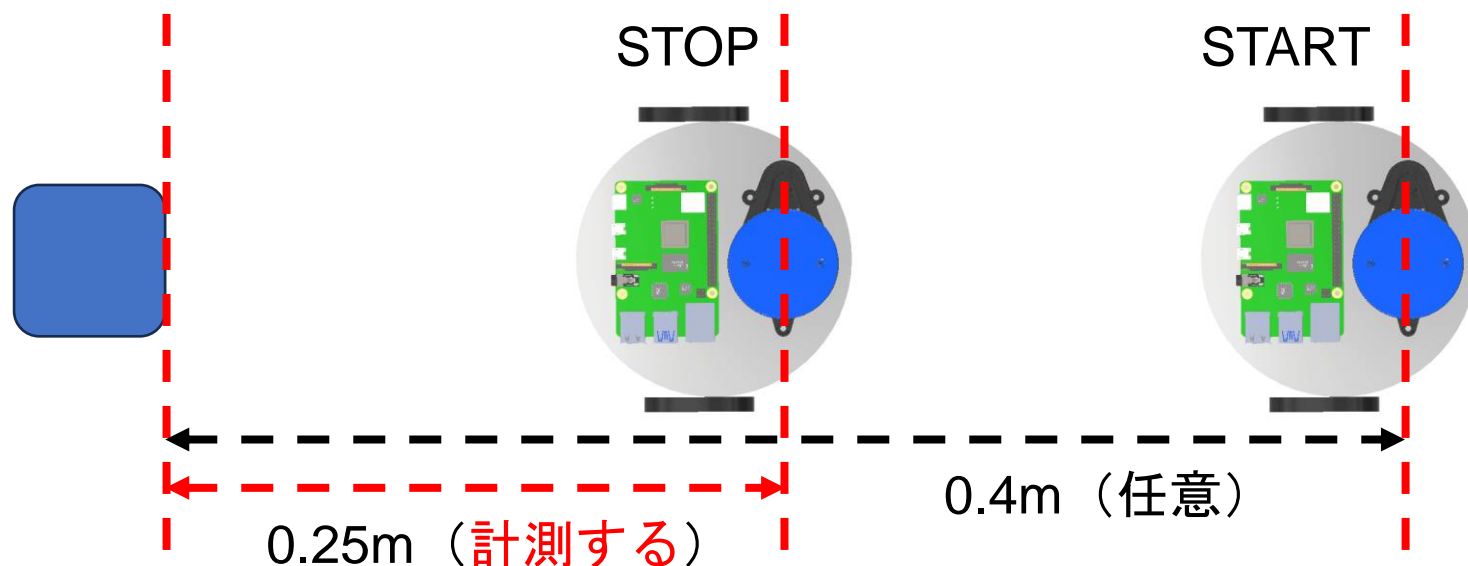
課題 2 のヒント

- /home/robot/catkin_ws/src/lightrover_ros/scripts/rover_controller.py を編集
- 関数 `callback_laser()` 内を編集
- rangesの配列を調べる
 - ranges[X] : LiDARの計測距離
 - X : 方向
- 課題1で関数controller()に追記した部分をコメントアウト
- roslaunch lightrover_ros controller.launch で実行可能

課題 3

ロボットの0.4m（任意）前方にオブジェクトを配置する．
LiDARの計測値を用いてオブジェクトから0.25mのところで停止
するプログラムを記述せよ．

停止実験を繰り返し（10回）行うことで，停止位置のばらつき
（標準偏差）を定規を用いて計測せよ．



ヒント（課題 3, 5, 6）

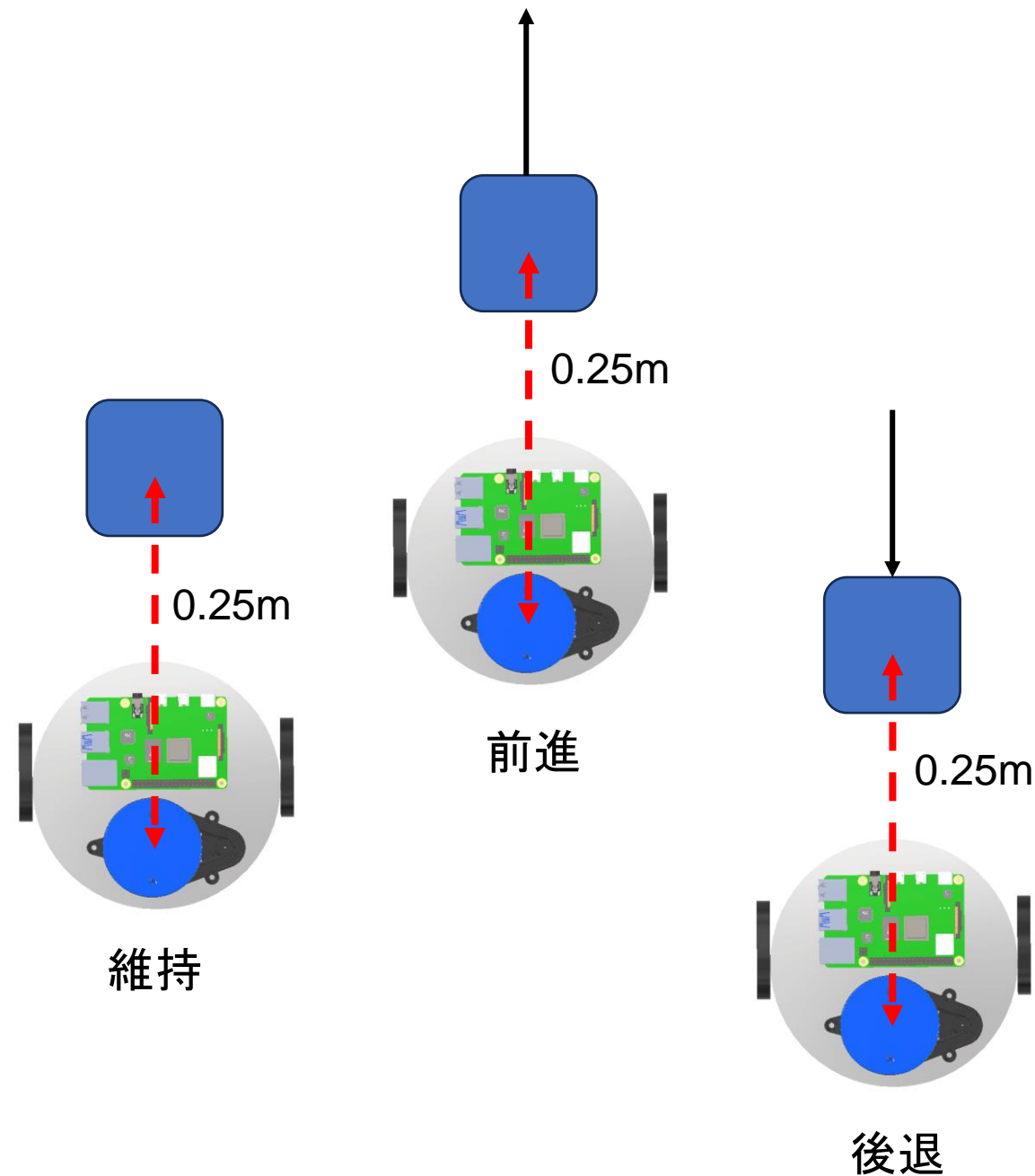
- `/home/robot/catkin_ws/src/lightrover_ros/scripts/rover_controller.py` を編集
- 関数 `callback_laser()` 内を編集
- `roslaunch lightrover_ros controller.launch` で実行可能

課題 4

課題 1 と課題 3 の結果を分析して，目標位置からずれる場合は改善するプログラムをそれぞれ記述せよ．

課題 5

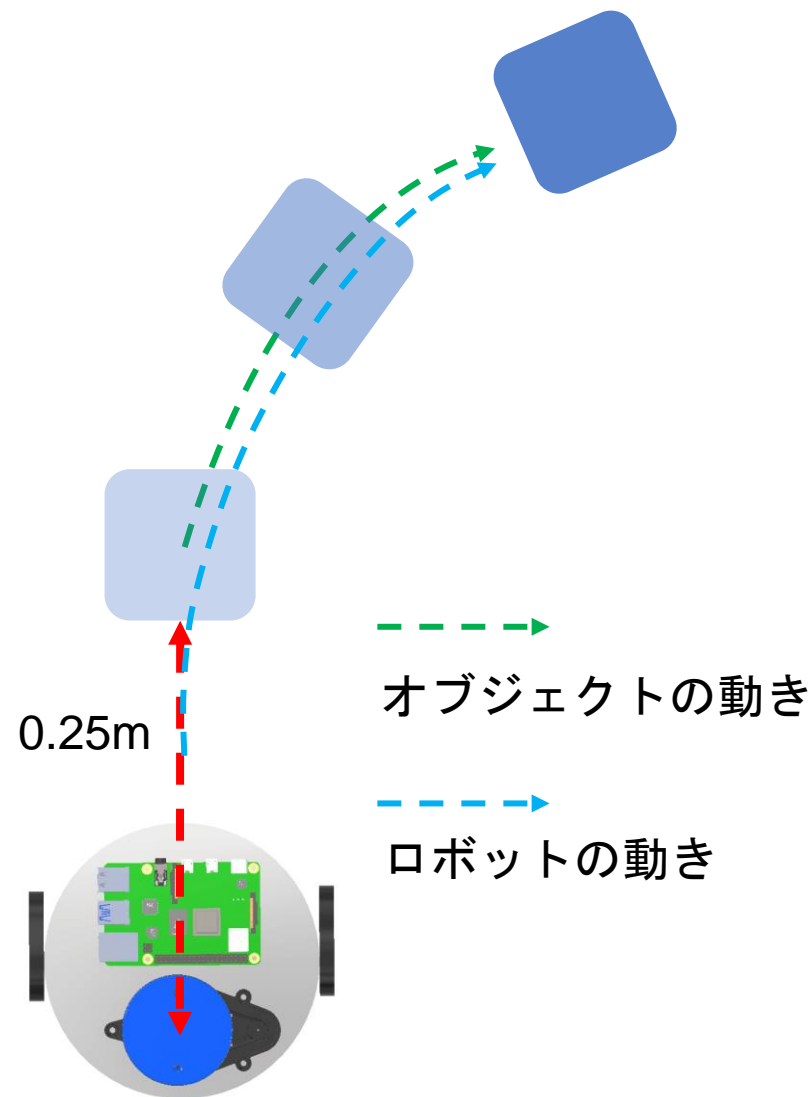
ロボットの前方にオブジェクトを配置する．LiDARの計測値を用いてオブジェクトから一定距離（0.25m）で追従するプログラムを記述せよ．



課題 6（追加課題）

課題5に関して，オブジェクトがカーブして移動する場合に対応させよ．ただし以下の条件のもと実装せよ．

- オブジェクトとの距離が0.5m以下なら0.25mを保つように動く
- オブジェクトとの距離が0.5m以上なら動かなくてよい



課題 6 のヒント

- カーブする場合は角速度の調整をする
- 配列中の最小値を探す（for文かPythonの組込関数rangeを利用）