自律運転トレーラー模型
概要

Saoto-Tech

[履歴]
Rev. 1.0：2020-03-23　　初版（土屋）

目次

1. はじめに

自動運転制御学習用トレーラーラジコン模型キットについて。

Saoto-Tech メール：saoto.tsuchiya@gmail.com

カラーは別途塗装しています。
（キットには、けん引するトレーラーは含みません。）

2. 資料

別冊解説書

（電子ファイルをノート PC の Document フォルダーに置きます）

No.	資料名	ページ数	最新 Ver.	日時	備考
1	模型改造方法	10	2.0	2020-03-20	
2	IF-Shield 仕様	3	2.0	2020-03-20	
2a	Arduino 仕様	4	2.0	2020-03-23	
3	OS セットアップ	8	2.0	2020-03-22	OS インストール
4a	ROS セットアップ１	5	2.0	2020-03-20	ROS の初期設定
4b	ROS セットアップ２	2	2.0	2020-03-23	自動走行用
5	操作方法	6	2.0	2020-03-23	

２～４における、インストール等のセットアップは既にされてあります。

１に沿って改造後、５に従って操作できます。

下記、上田先生の参考図書も同梱します。Ubuntu や ROS の使い方に関しては、この本が分かりやすいです。

参考文献・資料

①上田隆一「Raspberry Pi で学ぶ ROS ロボット入門」日経 BP 社

②小倉 崇（著）「ROS ではじめるロボットプログラミング」工学社

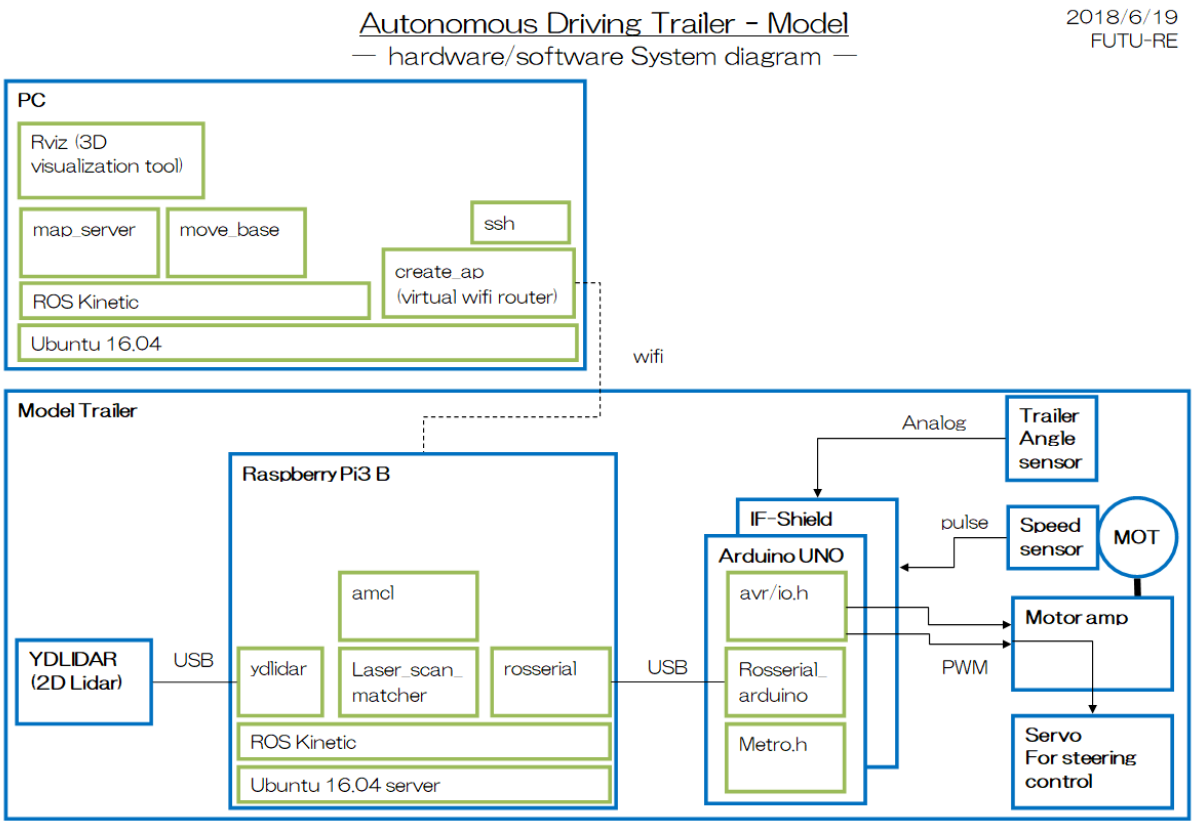
3. 内容物

項目		数量	備考
改造キット			
①②	Arduino UNO + I/F shield + case	1	組立済み
③	フォトエンコーダー	1	
③	取付治具	1	
③	取付ネジ　2.5 ㎜×6.0 ㎜	2	
④	エンコーダー用回転板	1	ジョイント用アタッチメントは 今回は使わないが、同封
⑤	Lidar 搭載アルミ板	1	加工済み
⑤	アルミ板	1	
⑤	プラバン	2	
⑤	10mm プラ柱＋ナット	4	
⑤	皿ねじ M3-8	8	予備含む
⑤	ねじ M3-6	5	予備含む Lidar 固定用
⑥	エンコーダー用配線	1	
⑥	フォグランプ用配線	1	
⑥	サーボ配線	2	
⑥	USB A-B	1	Arduino 用
⑥	USB A-microB	2	Raspberry Pi 電源用 Lidar 用
⑦	マジックテープ	2	
⑦	両面テープ	1	
購入物			
①	ラジコントレーラー模型 1/14 スカニア	1	
②	Lidar (RPLIDAR A2)	1	
③	Raspberry Pi 3 B	1	ケース、16G SD カード含む
④	モバイルバッテリー(10,050mAh, max2.4A)	1	
PC			
	DELL Vostro 14 5000(5490), Core i7, 16GB, 512GB SSD, マウス、外付け DVD スーパーマルチドライブ付属	1	インストール - Ubuntu 18.04 LTS - ROS - Virtual wifi

4. 制御システム（ハードウェア）

（1）システム図

メインの制御装置は車外 PC。車両には Raspberry Pi　と入出力用に Arduino を搭載。



（2）IF Shield

Arduino のインターフェースとして、IF-Shield を制作した。

別冊 2.　「IF-Shield 仕様」参照

5. 制御システム（ソフトウェア）

（1）システム図

上記、ハードウェアシステム図に併記。

（2）ネットワーク

PC と Raspberry Pi を wifi で接続。

PC 上のアプリケーション（create_ap)により、PC を wifi ルーターとして使用し、Raspberry Pi を接続している。

https://qiita.com/KuwabataK/items/5903c7584657151d576a

Arduino と Raspberry Pi は USB 接続し、ROS の ros_serial ライブラリを使用して、Arduino でも ROS トピックの配信／購読する。

（3）ROS

＜ノード＞

ROS ノードを Raspberry Pi と PC に分散して起動。Raspberry Pi と PC のそれぞれでタスク負荷が過剰にならないよう考慮。

Raspberry Pi

ydlidar_node ：YDLIDAR データを配信（YDLIDER を Raspberry Pi に接続しているので、Raspberry Pi で起動）

serial_node ：Arduino とのシリアル通信（Arduino を Raspberry Pi に接続しているので、Raspberry Pi で起動）

laser_scan_matcher_node ：Lidar データによるスキャンマッチング

amcl ：自位置推定

PC

roscore ：ノード管理（ROS マスターに設定している方で起動。現状、必ず PC からアクセスするので PC をマスターにしている）

map_server ：地図データの提供

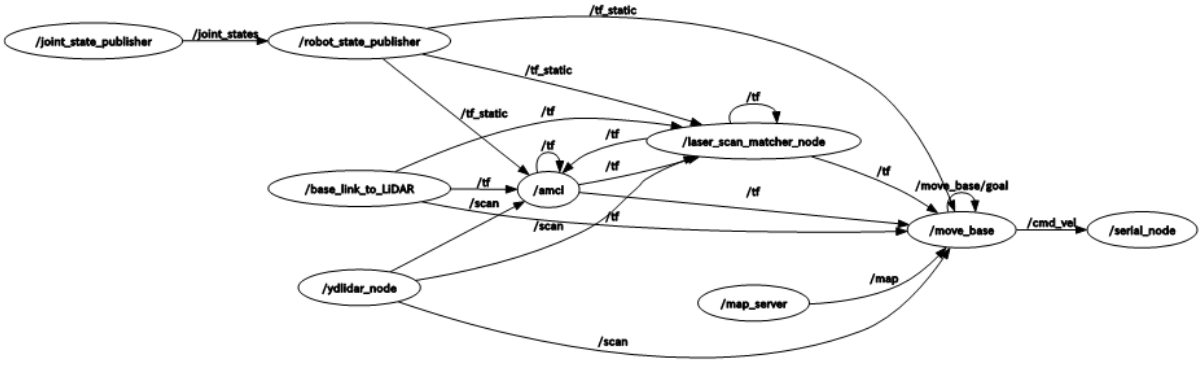
move_base ：経路計画

~~joint_state_publisher ：車両ジョイントの状態を配信~~

~~robot_state_publisher ：ジョイント状態に従い、車両内の tf 配信~~

Rviz ：GUI

ノード関係図（rqt_graph 結果）



＜座標系＞

今回のシステムでは、主に 4 つの座標系（map, odom, base_footripnt, base_link）とそれらの座標系をつなぐ座標変換 ft(transform)によって、車両の位置を表現している。

map ：静的地図に固定された座標系

odom ：オドメトリ（基本的には車速とステア角から積算される車両位置）の基準となる座標。車両初期位置を原点とする。

base_footprint ：車両座標を路面に投影した座標系

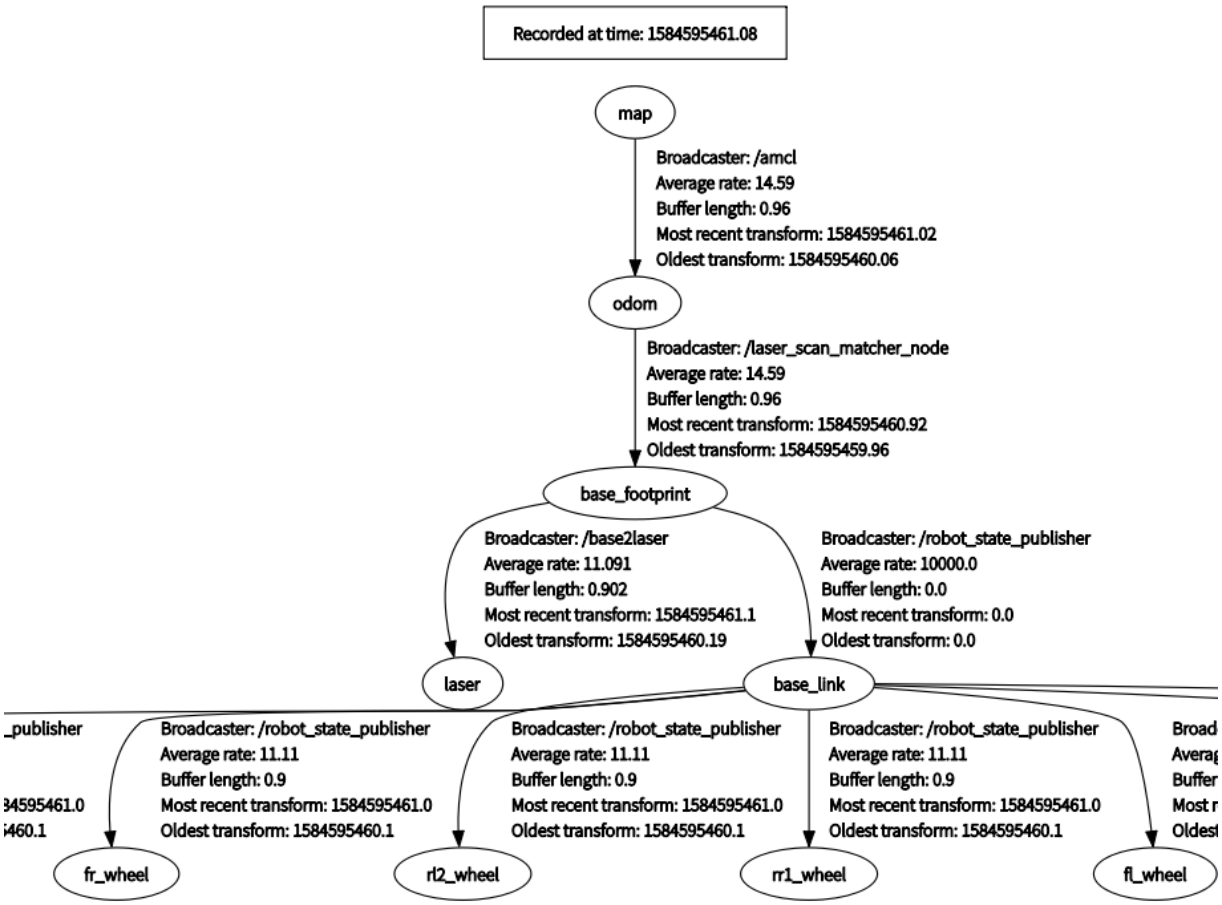
base_link ：車両座標。車両代表点を原点とし、前方を x 軸、上方を z 軸とする

base_link と base_footprint 座標間の座標変換は、本来は車両のピッチやロー運動及び上下動で変化するが、現状は固定とする。

base_footprint と odom 座標間の座標変換は、オドメトリに従う。

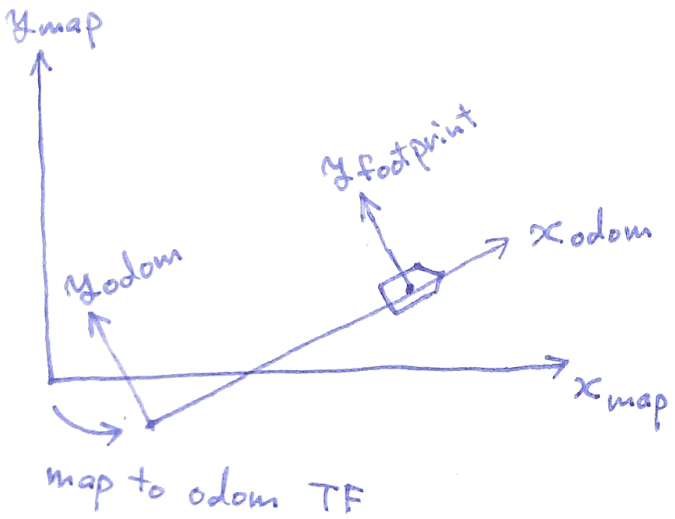
odom と map 座標の座標変換は、理想的には動かないはず。しかし、オドメトリの誤差により車両の位置が地図上でずれてくると、amcl によるマップマッチングによって odom 座標と map 座標の座標変換を補正するという方法で車両位置を修正する。

rqt_tf_tree の出力抜粋



（例）オドメトリでは直進しているつもりだが、少し左へカーブしている場合。

odom 座標では x 軸上だが、map 座標ではズレてくる。ズレは map 座標から odom 座標への座標変換 tf で補正。



＜Topic＞

おもな Topic（ノード間通信）

act_vel	：Arduino が配信する、車両情報（速度、ステア角度、トレーラーリンク角度）
scan	：Lidar スキャンデータ
odom	：オドメトリ（odom 座標での車両位置）
amcl_pose	：自位置（map 座標での車両位置）
cmd_vel	：Arduino への指令値（速度指令、ステア角指令）

（３）Arduino

別冊 2a。「Arduino 仕様」参照

課題

（１）Lidar の限界

2 次元タイプのため、一定以上の勾配変化があると地面や空を照射してしまい、自位置推定が不可能になる。

炎天下では検知不能になる。

（２）システム安定性

制御が止まることあり。要因として以下が考えられる

処理能力不足・・・CORE i7 の PC でも処理待ちになることあり。Raspberry Pi と分散処理することで改善はした。

WiFi 接続・・・車両と制御 PC 間を wifi が途切れた場合。

ラジコン信号受信・・・自動運転においてもラジコン送信機の信号が受信されないと駆動できないシステムになっている（この機能のおかげで、緊急時に停止が可能）。ので、ラジコン送信機信号が常に届くように留意が必要

自己位置推定エラー・・・自己位置推定に失敗すると、制御不能になる。Lidar 能力の限界には注意が必要

（３）自己位置推定法の限界

以下のような状況では自己位置推定が失敗することあり

炎天下や勾配変化があると、Lidar 能力の制限により

壁などの目標物が、遠かったり数が少ない場合。キャビンを取り外して、360°視野にすると改善はする

高速走行

（４）車両挙動の不安定性

まだスムーズに走行できず。以下の要因が考えられる

処理待ちで制御が遅れた場合

パスプランニングの適合不足か

車両制御マイコンの遅れ・精度不足・・・改善の余地もあるが、遅れを考慮した制御が必要か

駆動モータ及びステアリング機構の性能不足。特にステアリングサーボのトルクやステアリング機構の剛性が不十分に感じられる

速度制御の性能・・・Arduino で簡易的な速度フィードバックをしているが、改善の余地あり

障害物検知・・・回避行動によりギクシャクしてしまう

切り返し・・・曲がれないと判断して、切り返しを多発してしまう

自己位置推定法

下記に示した１～４の方法を試した。現状は３と４を連携して自位置を推定している。

座標の関係

/map（固定された地図座標）

↓amcl が位置関係を補正

/odom（scan_matcher の原点）

↓laser_scan_matcher が推定する車の移動量

/base_footprint（路面上の車両基準点）

（１）オドメトリ

Arduino で算出した車両速度とヨーレート（旋回速度）から、車両移動量を計算する。

adt_odom.py に実装。

入力：

・車両速度 v

・ヨーレート γ

出力：

・車両位置 x,y（odom 座標）

・車両の向き θ（odom 座標）

$x = x + v * \cos\theta * dt$

$y = y + v * \sin\theta * dt$

$\theta = \theta + \gamma * dt$

※現状、Arduino で γ が大きめに算出されているようなので、ここで小さめに補正している。本来は Arduino 側を修正するべき。
γ はステア角から計算しているが、そもそもステア角を計測しておらず、サーボへの指令値∝ステア角としているので、あまり精度は望めない。

（２）hector slam
上記オドメトリの精度が悪いので、hector slam ライブラリを利用した車両の移動量計算を試す。
hector_slam は、オドメトリ情報を使わずに、スキャン情報のみで車両の移動を推定する。地図作成にもこのライブラリを使用しており、精度は良いはず。
http://wiki.ros.org/hector_slam

オドメトリ情報を使用せず、Lidar によるスキャン情報のみで車両の移動を推測するため、失敗するとあらぬところまで車両を飛ばしてしまうことがある。
ネット上の hector のデモでは、たいがい 40Hz スキャンの Lidar を使っているが、我々の Lidar は 10Hz しかない。素早い動きは無理があるのかもしれない。

（３）Laser Scan Matcher
オドメトリ情報を考慮すれば位置が飛ぶことはないはずと思い、laser_scan_matcher を試す。
laser_scan_matcher は、オドメトリ情報を参照しつつ、スキャン情報で車両移動量を推定。
http://wiki.ros.org/laser_scan_matcher

良好に自己位置推定できるが、多少車両位置が飛ぶことがある。目標物が遠かったり・少ない場合、視野角が少ない場合は、どうしようもないのかもしれない。
laser_scan_matcher をオドメトリ情報を使わない設定にしたところ、１番安定している。当初の思いとは異なるが、現状この方法を用いている。

hector はスキャン情報がおかしくなると１番正しいと思われる場所へ飛ばす感があるが、
laser_scan_matching はそういう時には移動させないようである。今回は、最終的に amcl で地図とのマッチングをして絶対位置の補正をするので、hector よりも Laser_scan_matching の方が相性がいいようである。

（４）amcl（適応型モンテカルロ位置推定法）
地図上の自己位置を推定するライブラリ
オドメトリ（車速とステア角から計算した車両位置）を基準に、地図とスキャン情報（Lidar 出力）のマッチングで自己位置を推定。
http://wiki.ros.org/amcl

今回は上記のように、オドメトリとして laser_scan_matcher 出力を使用。

地図マッチングには、地図が正しいことが重要。物の配置が変わった場合は、地図を作りなおすべき。

現在の主要パラメーター設定

odom_model_type = diff-corrected ：バグフィックスした、非ホロノミックタイプ

kld_err = 0.01 ：真値と推定値との誤差。積極的に補正うるよう小さく設定

initial_cov_xx, yy, aa = 0.01, 0.01, 0.01 ：ロボットの初期姿勢の分散．値が大きくなるほどばらまくパーティクルの分布，角度が広がる。初期位置は基本手動で合わせるので、勝手に移動してしまわないよう小さく設定。