自動運転車両
Arduino 仕様

FUTU-RE

[履歴]
Rev. 1.0：2018/6/20 初版(土屋)
Rev. 2.0：2020/03/23　改訂(土屋)

1. はじめに

ADT のインターフェースとして搭載する Arduino の仕様書。
ROS_Serial を使用し、Raspberry Pi に対して ROS ノードとしてふるまう。
PC から USB によるシリアル信号で要求等を受信
PC へ USB によるシリアル信号で速度情報等を送信
速度パルス入力から速度計算
ステアサーボ指令 PWM 出力
モータ駆動指令 PWM 出力

ソフトの変更は必要ないが、以下の適合は車両ごとに必要かもしれない。
・ステア角要求に対する、ステアリングサーボへの指令値
・モータ速度制御の定数

ソースコードは、以下に置いてあります。
https://github.com/saoto28/auto_drive_model/tree/master/arduino/ADT_speed_control_ros
及び、PC の~/catkin_ws/src/auto_drive_model/ arduino/ADT_speed_control_ros

2. 入出力

Subscribe（受信）1：cmd_vel　← geometry_msgs/Twist
　geometry_msg/Vector3 linear
　　float64 x　：spdVehReq・・・要求速度[m/s]
　　float64 y
　　float64 z
　geometry_msg/Vector3 angular
　　float64 x
　　float64 y
　　float64 z　：angSpdReq・・・要求角速度[rad/s]
　0.1s 以上受信されなければ、モータ指令停止

Publish（送信）1：act_vel　← geometry_msgs/Twist
　geometry_msg/Vector3 linear
　　float64 x　：spdVehAct・・・実速度[m/s]
　　float64 y　：spdReq・・・要求速度（デバック用）

float64 z ：cmd_mot・・・モータコマンド（デバック用）
geometry_msg/Vector3 angular
float64 x ：angJointTrailer・・・トレーラ連結角度[rad]
float64 y ：angReq・・・要求ステア角（デバック用）
float64 z ：angSpd・・・ヨーレート[rad/s]
0.1s 毎に発信

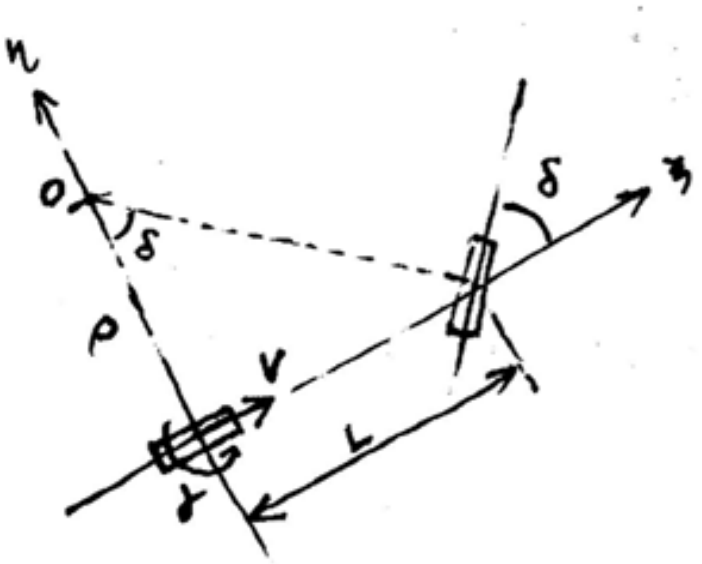
※ act_vel で速度だけ出力し、オドメトリの計算は Raspberry Pi（または PC）でさせる。

3. ステア角指示
車速 v[m/s]を後輪速度とすると
車両ヨーレートγ[rad/s] = v/ρ = v/L × tanσ

tan$\sigma \fallingdotseq \sigma$[rad] とおけば

$\sigma = \gamma L/v$

として、要求角速度（ヨーレート）から目標舵角が算出される。



ここで、ホイールベース L の後方起点は後輪に平行に進む点なので、後輪が４輪の場合は４輪の中心点でよいだろう。
または荷重がかかる点ということで、トレイラー連結軸でもよいかと思うが、どちらにせよほぼ同じ位置。

車両停止時～低速時は、この単純計算では大舵角になりやすく、実車では違和感が生じる恐れがある。が、模型においてはこのままでもよいのではないだろうか。
その方が、低速時でも確実に要求に沿った旋回をしやすくなる。

現在、目標舵角計算において車速として目標車速を使っている。そのため、実際には速度が出ていない場合に舵角を小さく見積もってしまいがちのようである。
発進時に確実にステアリングを切るために、実車速を考慮する必要があるかもしれない。
（その場合、加速時に過度にステアリングを切るおそれもあるが）

4. FailSafe
(1) Raspi と通信が途切れた場合、モータ出力を止める

5. rosserial

Arduino を ROS ノードとして扱うためのツール

ROS の rosserosrunrial パッケージのインストール
```
sudo apt-get install ros-kinetic-rosserial ros-kinetic-rosserial-arduino
```

Arduino SDK に rosserial ライブラリの追加
```
cd ~/sketchbook/libraries
rosrun rosserial_arsuino make_libraries.py
```

使用時は、接続する PC をまたは RasPi 側で、通信用ノードを立ち上げる。
```
rosrun rosserial_python serial_node.py _port:=/dev/ttyACM0
```

Navigation が出力する raw_cmd_vel を cmd_vel として arduino へ送る場合は、
```
rosrun rosserial_python serial_node.py _port:=/dev/ttyACM0 ¥
/cmd_vel:=/navigation_velocity_smoother/raw_cmd_vel
```

テスト用、PC からキー操作で cmd_vel を送るノード
```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

6. Metro

Arduino で一定周期のタスクを模擬するため、Metro ライブラリを使用。

Arduino 開発環境に Metro ライブラリーを追加する必要あり。
下記のフォルダをライブラリーフォルダに置くか、圧縮ファイルのまま開発環境からライブラリの追加をすればよいはず。
https://playground.arduino.cc/Code/Metro

7. 速度計測

プロペラシャフトに取り付けたエンコーダーのパルス周期から速度を計算。

パルスの立下り割込みで周期を計測。（リンギング防止でセンサ出力にコンデンサーを付けているが、立ち上がりは過度になまされている。立下りのほうが影響が少ないようなので）

簡易的な IIR ローパスフィルタでフィルタリング。
計測周期 x(n)に対し、
フィルタ値 y(n) = 0.2 * x(n) + 0.8 * y(n-1)

上記フィルタ後のパルス周期： λ [s]
エンコーダースロット数：N=24

デファレンシャルギヤ比：RAT_Diff = 15/40
タイヤ半径：RADIUS=41mm

速度 V=RAT_DIFF＊2π*RADIUS／(N＊λ)

速度計算周期 100ms 間パルスが検出されない時（40mm/s 相当）は、車速 0 とする。

Arduino UNO の時間計測分解能：4us
100km/h でもパルス間隔は 145us あるので、分解能に問題はないだろう。

8. 速度制御
cmd_vel トピックで指令された要求速度に対し、速度をフィードバックしモータを速度制御する。速度制御には、フィードバックとフィードフォワードを足し合わせた 2 自由度制御とし、安定性を確保。フィードバックは PI 制御。

タミヤのモータコントローラ MFC-01 の仕様として、前進後 1 回目の後進指令は無視され、後進は 2 回目以降でしたモータ駆動がされない。
従って、前進後の後進時は、後進指令を 2 回出す処理を入れる。

9. PWM 出力
Arduino の標準関数として Servo 用パルス出力関数があるが、今回のタミヤ MFC-01 はこの関数の出力パルスを受け付けなかった。調べると、servo()関数の出力周波数 50Hz に対し、タミヤの受信機が発しているパルス周波数は 73Hz 程度であった。
今回、servo()関数の代わりに、マイコン（ATmega 328P）のレジスタを直接いじり 73Hz でパルスを出力することで、モータコントローラを操作している。

参照：http://blog.kts.jp.net/arduino-pwm-change-freq/

ステアリングサーボ指令は 50Hz でも反応するが、モータに合わせて 73Hz としている。
ちなみに、出力電圧も、Arduino は 5V、受信機は 3V と異なるが、これは問題ではなさそうだったので、そのままにしている。（分圧したほうがよかったかもしれない）

パルス幅 0.5～1.5～2.5ms に対しサーボは以下のように反応
モータ：前進～停止～後進
ステアサーボ：左～直進～右