Vehicle Control Unit – Software Algorithms

Table of Contents

[IMU Correction 3](#_Toc194403344)

[ECUMaster GPStoCAN V2 Correction 3](#_Toc194403345)

[Pitch and Roll Correction 4](#_Toc194403346)

[Position Correction 4](#_Toc194403347)

[IMU Accelerations 5](#_Toc194403348)

[CG accelerations 5](#_Toc194403349)

[Centripetal acceleration from angular velocity 5](#_Toc194403350)

[Angular acceleration from vehicle angular acceleration (Euler acceleration) 5](#_Toc194403351)

[Coriolis acceleration 6](#_Toc194403352)

[Position correction equation 6](#_Toc194403353)

[Velocity and Angular Velocity Correction 6](#_Toc194403354)

[Vehicle Side-slip angle from IMU Heading 7](#_Toc194403355)

[Reference Velocity Components 8](#_Toc194403356)

[Yaw Acceleration (TODO) 9](#_Toc194403357)

[Wheel Load Calculation (needs improved) 10](#_Toc194403358)

[Lateral Load Transfer 10](#_Toc194403359)

[Longitudinal Load Transfer 11](#_Toc194403360)

[Tire Static Normal Loads 11](#_Toc194403361)

[Model representation of normal loads 11](#_Toc194403362)

[Slip Ratio Calculation 13](#_Toc194403363)

[Tire Ground Velocity 13](#_Toc194403364)

[Slip Ratio Calculation 14](#_Toc194403365)

[Track Width Migration (TODO) 16](#_Toc194403366)

[Yaw Moment Error Per Tire 16](#_Toc194403367)

[APPS Sensors 17](#_Toc194403368)

[APPS Plausibility Checking 18](#_Toc194403369)

[APPS Power Percentage 18](#_Toc194403370)

[Brake Pressure sensors 19](#_Toc194403371)

[Gauge Pressure adjustment 19](#_Toc194403372)

[Pedal Hysteresis 19](#_Toc194403373)

[Pedal Calibration 20](#_Toc194403374)

[Brake Bias Adjustment 20](#_Toc194403375)

[Brake Pressure Sensor Plausibility 20](#_Toc194403376)

[Brake Pressure and Pedal Percentage 21](#_Toc194403377)

[Instantaneous Maximum Power Limit 22](#_Toc194403378)

[Motor Power Limits (TODO) 22](#_Toc194403379)

[Inverter Power Limits (TODO) 22](#_Toc194403380)

[Battery Power Limits (TODO) 22](#_Toc194403381)

[Tire Longitudinal Traction Limit (TODO) 23](#_Toc194403382)

[Control System References 24](#_Toc194403383)

[Yaw Rate Reference 24](#_Toc194403384)

[Longitudinal Acceleration Reference 24](#_Toc194403385)

[Torque Optimization 25](#_Toc194403386)

[Multi-Objective Constrained Optimization Problem 25](#_Toc194403387)

[Decision Variables 25](#_Toc194403388)

[Weighted-Sum Method 25](#_Toc194403389)

[Constraints 25](#_Toc194403390)

[Yaw Moment Objective Function 26](#_Toc194403391)

[Longitudinal Acceleration Objective Function (TODO) 27](#_Toc194403392)

[Slip Angle Objective Function (TODO) 27](#_Toc194403393)

[Linearization 27](#_Toc194403394)

[Tire Force Linearization 27](#_Toc194403395)

[Conversion into a Convex Quadratic Programming Problem 28](#_Toc194403396)

[CVXGEN Solver 29](#_Toc194403397)

[Converting Objective Functions into P and q matrices 29](#_Toc194403398)

# IMU Correction

## ECUMaster GPStoCAN V2 Correction

The torque vectoring algorithms use ISO 8855 for the global vehicle coordinate system.

A screenshot of a computer

Description automatically generated

*ISO 8855 vehicle coordinate system*

A diagram of a car

Description automatically generated

*ECUMaster GPS to CAN V2 coordinate system*

To avoid confusion, the IMU is mounted in the car per the orientation described on the enclosure. In software, the Y and Z axis must be inverted. Additionally, the IMU acceleration output is in Gs, and angular velocity is in degree / sec:

Assumption: yaw rate for IMU is positive for counter-clockwise motion, in the direction of positive z axis; therefore it should be reversed if the axis is getting flipped

## Pitch and Roll Correction

as used in the double track model are oriented relative to the ground, not the vehicle. As the vehicle pitches and rolls, sensor measurements need to correct to get measurements relative to the ground orientation.

If the roll gradient for the vehicle is , the IMU lateral acceleration error is:

A graph of a graph

Description automatically generated with medium confidence

Even at roll gradients higher than design, the error is under 0.6%, for pure roll. It will be higher with combined roll/pitch movement. Because the IMU does not output roll and pitch angles, a separate accelerometer/gyro/magnetometer with sensor fusion is needed to estimate this. For now, we will not correct for this.

## Position Correction

We care about accelerations at the center of gravity. If the accelerometer needs to be placed in a location different from the center of gravity, we can correct its output to get C.G. accelerations.

We assume the IMU axis still align with the vehicle axis. However, when the vehicle has a rotational velocity (mainly yaw velocity), the accelerometer will read an acceleration from this when not placed at the C.G. We can assume that only X and Y accelerations need to be corrected.

### IMU Accelerations

The accelerometer will read a superposition of vehicle C.G acceleration, centripetal acceleration, angular acceleration, and Coriolis acceleration:

### CG accelerations

If the IMU and vehicle coordinate systems are aligned, the vehicle CG accelerations directly appear in the IMU acceleration without any rotational transformation:

### Centripetal acceleration from angular velocity

Occurs along the vector from IMU position to C.G. location. Centripetal acceleration can be expressed in terms of angular velocity (rad/s), and the distance is the magnitude of the IMU position vector:

Which can be expressed as accelerations in the vehicle axis:

### Angular acceleration from vehicle angular acceleration (Euler acceleration)

To have angular velocity, the vehicle must have an angular acceleration, which will induce accelerations into the IMU separate of the centripetal accelerations. There will be a tangential acceleration at the IMU in the direction of angular acceleration:

### Coriolis acceleration

Occurs in the vector from IMU position to C.G. location. It is quite small so often assumed to be zero to simplify correction.

### Position correction equation

Accurate correction requires good measurement of yaw velocity and yaw acceleration.

## Velocity and Angular Velocity Correction

Assuming the IMU axis and vehicle axis are aligned, linear and angular velocities will be measured as they are at the C.G, no correction required. Derivations of these to get yaw accelerations also does not require correction, therefore these readings from the IMU can correct IMU acceleration data.

# Vehicle Side-slip angle from IMU Heading

The ECUMaster GPS to CAN V2 gives both vehicle X axis heading angle and the heading angle of the vehicle motion, in degrees from 0 – 360.

Assumptions (need to test IMU to verify)

* IMU references headings based off an origin heading determined at initial power up
* IMU heading angles increase in counterclockwise direction
* IMU datasheet seems to imply it can output 0 or 360 degrees for the same heading

To correct for the 0 / 360 degree issue, convert to radians, and to limit the range of heading angles to , we adjust the IMU output first:

Vehicle side-slip angle can then be calculated as:

for

# Reference Velocity Components

The ECUMaster GPS to CAN V2 gives ground speed in kph and heading motion.

# Yaw Acceleration (TODO)

Yaw acceleration is the derivative of yaw rate and not directly measured. A fast algorithm for numerical derivative estimation should be developed for use on the embedded firmware.

* Taylor series
* Reimann derivative
* Central limit theorem

Verify timing with GPIO toggling via atomic register write, or using trace

# Wheel Load Calculation (needs improved)

We can use vehicle accelerations around the C.G. to calculate load transfer and determine vertical tire loads.

– longitudinal acceleration at C.G.

– lateral acceleration at C.G.

– vertical acceleration at C.G.

– vertical height of C.G. position from ground

– distance from C.G. position to front axle

– distance from C.G. position to rear axle

– total vehicle mass, with driver

– gravitational acceleration, 9.81 m/s^2

– track widths

– wheelbase

## Lateral Load Transfer

Total lateral load transfer from right tires to left tires (track widths are averaged):

Distribution of lateral load transfer between front and rear wheels:

A math equations with numbers and symbols

Description automatically generated with medium confidence

Note: See RCVD book pg 683. These are the simplified equations of the assumed steady-state values. They probably won’t be accurate during transient maneuvers which is not desirable.

These may not be accurate enough for what we are trying to achieve. Alternative approaches include:

* Distribution of TLLT to front and rear wheels using a roll stiffness magic number derived from spring and roll bar calculations
* putting push rod load sensors on each wheel and more directly estimating wheel loads
* passing C.G. accelerations to a transient kinematics model built in Simulink that can estimate lateral load distribution for both transient and steady state

## Longitudinal Load Transfer

Total longitudinal load transfer from front tires to rear tires:

A\_x is in [g].

## Tire Static Normal Loads

Tire Normal Loads, assuming equal weight distribution laterally:

## Model representation of normal loads

The plant model used to simulate the dynamics must calculate the normal loads on each wheel accounting for both load transfer and tire lift off the road. Normally, a kinematics model representing the independent suspension, spring rates, and torsional chassis stiffness is used to calculate the normal loads, however the plant model will include a simplified representation of load transfer without the transient effects (at the moment). The model must consider the following constraints:

* The normal load on a tire must never go negative
* The sum of the normal loads must equal the total mass \* 9.81 m/s^2
* Lateral load transfer must use a constant lateral load transfer distribution percentage
* When on two wheels, the distribution of normal force should represent the static weight distribution across those axles
  + For example, lifting on two wheels when cornering should not result in the loading of the two outside wheels based on the TLLTD but rather the static weight distribution of the vehicle
  + Lifting the front / rear wheels when braking would also require that if there is any lateral load transfer, it can only be between the two tires on the ground and therefore the TLLTD ratio also does not apply

# Slip Ratio Calculation

Slip ratio is the estimate of the sliding between the tire surface and the ground at the tire. It is used for calculating the longitudinal force a tire is producing.

## Tire Ground Velocity

The ground velocity at the tire is the reference velocity of the vehicle and an additional tangential surface velocity component caused by the yaw angular velocity:

The tire distance vectors are:

The ground velocity vectors in vehicle reference frame are:

simplified:

The slip ratio is calculated as used in the FSAE TTC data. It uses the road speed, which is the ground velocity in the direction of the vector that is made with the slip angle and the center of the tire, in the tire’s local X axis reference frame:

A diagram of a rectangular object with red lines and arrows

Description automatically generated

The ground surface velocities in the tire coordinate frame are oriented relative to the slip angle and not the longitudinal orientation of the tire.

## Slip Ratio Calculation

Because we are using FSAE TTC data captured at Calspan TIRF, we will use the TIRF definition for slip ratio, which uses the effective radius of the tire (and not loaded radius).

* : angular velocity of the wheel in rad/s
* : surface velocity of the wheel over the ground

The typical definition of V includes a cosine factor for slip angle, however that and yaw rate effects are accounted for in the calculation of . is found at the free rolling, SA = 0 deg condition, however it varies as a function of velocity, inclination angle, and tire normal load:

# Track Width Migration (TODO)

As we load up each tire, the sidewall flexes. The center of pressure for the tire forces will migrate, possibly up to 1/3 of the rim width. Estimating track width migration involves using the overturning moment Mx.

## Yaw Moment Error Per Tire

A graph of a graph with a colorful rectangle

Description automatically generated with medium confidence

In a lateral cornering condition, torque vectoring moments are applied by driving the motor to an Fx to achieve a target yaw moment per tire. With expected track widths and contact path deflection (approximately max 1/3 of tire width), there could be as much as 10% error between the expected yaw moment contribution of a tire and what is actually occurring. This error subtracts from the yaw moment for outside tires and adds for inside tires. Therefore this is something that should be corrected for using tire model data.

# APPS Sensors

Two APPS signals are read from the ADC on the microcontroller and then checked against each other to validate that the signal is plausibility and can be used to safely control the vehicle.

A graph with a line and a line

Description automatically generated

A white text with black text

Description automatically generated

Algorithm must detect open circuit, short to ground, short to sensor power, and short between signal lines. The algorithm must also allow calibration of the accelerator pedal stops to correlate the physical travel of the pedal with the torque request output.

A pull-down is implemented on the sensor signal to ensure that an open circuit puts 0V on the ADC input.

## APPS Plausibility Checking

All values are expressed in raw ADC counts.

The voltage limits in the plausibility checks are converted to raw ADC counts as such:

* n-bit ADC
* AVDD voltage: 3.3V

## APPS Power Percentage

If the APPS are plausible, the APPS ADC counts are converted to a power percentage that is passed to the control algorithm. There is a low pedal dead zone percentage that ensures the car does not “inch” forward due to sensor noise when the pedal is fully released. There is also a high pedal dead zone percentage that helps the driver maintain 100% acceleration without worrying too much about keeping the pedal fully traveled.

# Brake Pressure sensors

P/N: M2021-000005-2K5PG

* 5V supply
* Output: analog 0.5 to 4.5V ratiometric
* Gauge pressure output
* Pressure Range: 0 – 2500 PSI (gauge)

A graph with a line

Description automatically generated

A white text with black text

Description automatically generated

## Gauge Pressure adjustment

The sensors output gauge pressure, which is a pressure relative to atmosphere pressure. No adjustment is needed since the net force applied to the brake calipers is a function of gauge, not absolute.

## Pedal Hysteresis

Hydraulic systems are incompressible, so there is no force to push the pedal and caliper pistons back except any springs incorporated into the master cylinder or the pedal design. Therefore, there can be lingering brake line pressures present even when the pedal is not pressed. You can graph with recorded data by making a scatter plot of brake line pressure vs vehicle Ax when Ax is negative.

Pedal dead zones need to be comfortably above the maximum hysteresis pressures otherwise these pressures can be interpreted as driver inputs in the regen braking / torque vectoring algorithms.

## Pedal Calibration

The minimum and maximum pressure is calibrated by having the driver pressing the pedal as hard as they can and recording the value. This allows a brake percentage to be determined and allows a dead zone for brake light and regen braking to be correlated to brake pressure outputs.

The values calculated here are in raw ADC counts as such:

* n-bit ADC
* AVDD voltage: 3.3V

is the calibration value when the driver presses the pedal as hard as possible:

is the hysteresis value when the driver is off the pedal:

is the minimum ADC reading required to trigger the brake light:

## Brake Bias Adjustment

Brake bias adjustment changes the maximum pressure both circuits have when the driver presses the pedal. Adjusting it while driving can mess up some of the plausibility checks so calibration should be re-done after setting it.

## Brake Pressure Sensor Plausibility

Plausibility checks are done for front and rear circuit sensors independently. Out of range of the calibration limits does not imply an implausible situation, just bad calibration (hysteresis during calibration, or the driver didn’t press hard enough).

The voltage limits in the plausibility checks are converted to raw ADC counts as such:

* n-bit ADC
* AVDD voltage: 3.3V

An implausible situation for brake pressure sensors does not require the VCU to open the shutdown loop, but an error flag should be sent over the CAN bus. It would mainly be used to determine if the BPS input can be used for regen / torque vectoring control

## Brake Pressure and Pedal Percentage

# Instantaneous Maximum Power Limit

See Simulink model of maximum power limit calculation.

The maximum power limit is calculated during the torque distribution section of the control algorithm. It is used as an input to ensure that the individual wheel torques calculated to meet the yaw moment output from the controller don’t exceed rules limitations, driver pedal input, or physical limitations of the motors, inverters, or battery systems.

Note: this value is not relevant to the power limit imposed by vehicle traction.

## Motor Power Limits (TODO)

Each individual motor power limit is a function of speed, temperature, voltage, and field weakening

## Inverter Power Limits (TODO)

Inverter power limit is a function of temperature. This could be something sent from the inverter to the VCU via CAN bus.

## Battery Power Limits (TODO)

The battery discharge power limit is a function of the pack voltage and discharge current limit, where the discharge current limits could be temperature dependent. The power limit for the battery should be reduced as we approach the minimum pack voltage, as high current will cause a voltage drop that can cause the BMS to trip before the pack is fully discharged.

# Tire Longitudinal Traction Limit (TODO)

Before we can take the actuating yaw moment that is output from the TV controller and distribute it into motor torques, we need to find the range of possible tire long force for each tire. This then becomes a hard limit that is factored into the torque distribution optimization

For each tire:

1. Estimate tire conditions (normal load, slip angle, camber, pressure, etc)
2. Sweep slip ratio at that condition and grab the Fx curve
3. Hard limits are min and max of that curve

[fx, slip\_ratio] = find\_fx\_curve(tire\_fz, tire\_sa, tire\_ia, tire\_p)

tire\_fl\_fx\_curve = find\_fx\_curve(tire\_fl\_fz, tire\_fl\_sa, tire\_fl\_ia, tire\_fl\_p)

[min, max] = find\_traction\_limit (tire\_fx\_curve)

tire\_fl\_fx\_limits = find\_traction\_limit (tire\_fl\_fx\_curve)

1. When a torque distribution is determined, this same curve can then be used for determining a reference slip ratio to be input into the slip ratio controller

SL = find\_slip\_ratio(target\_wheel\_torque, fx\_curve)

# Control System References

## Yaw Rate Reference

## Longitudinal Acceleration Reference

# Yaw Moment Controller

## Gain Scheduling

## Tuning PI gains

The Simulink model is non-linear, SISO, with feedforward disturbances. Typically, the model would get linearized at multiple operating points, with linear state-space models developed to create a linear-parameter-varying (LPV) system. Then, at each operating point (combination of initial velocity and steering wheel magnitude), each linear system would be independently tuned to create a lookup table of PID gains, which get assigned to the control in real-time. However, the feedforward disturbances make linearization and tuning with the typical PID Tuner or Control System Tuner MATLAB apps more complicated.

A more direct approach was chosen, in which the Design Optimization Toolbox was utilized. This app uses a gradient decent approach to directly tune any parameter of the non-linear model to specific response characteristics, including PID gain variables. Base workspace variables for Kp and Ki gains were created, and the Response Optimizer app was used to optimize the step response by varying design variables Kp and Ki. Kd is left out because it can amplify signal noise

The following step response envelope was targeted:

|  |  |
| --- | --- |
| Rise Time (s) | 1s |
| Settling Time (s) | 2s |
| Percent Overshoot | 10% |
| Percent Rise | 80% |
| Percent Settling | 1% |
| Percent Undershoot | 1% |

The final steady-state value was adjusted to match the reference yaw-rate for each operating point. It should be noted that this desired step response was arbitrarily chosen, and significant aspects such as percent overshoot could be adjusted based on driver feedback.

## Mz Saturation

When tuning the gains please note that the limiting factor is the vehicle’s ability to achieve the Mz actuating signal via the torque optimization algorithm. Since we are tuning the PID controller via bypassing the torque optimization block and applying Mz directly to the vehicle CG, this physical limit must be accounted for by saturating the Mz actuating signal with an upper / lower maximum bound.   
  
Once torque optimization is added, we can see the real saturation limit of the vehicle and adjust our Mz saturation limit with a better approximation. I don’t think it is realistic to auto-tune with the TD block included (may cause fault or take forever to

A screenshot of a computer

Description automatically generated

*Using respone optimizer app to tune Ki and Kp for a given operating point.*

# Torque Optimization

## Multi-Objective Constrained Optimization Problem

We have three objective functions to optimize:

1. Meet yaw moment requirement from high level yaw controller
2. Meet longitudinal acceleration requirement from APPS sensor
3. Minimize slip ratios

### Decision Variables

The decision variables are the set of inputs into the cost function which we want to optimize. In this case they are the motor torques , where for

### Weighted-Sum Method

where

* are objective functions, with n number of objectives.
* are weights. The weights typically sum to 1 so that they can be easily described as percentages.
* are scaling factors which normalize each objective function. Note that is the true weight of each function, but we break it up to make relate to something meaningful.
* are constraints with minimum l\_i and maximum u\_i

### Constraints

The constraints are

|  |  |
| --- | --- |
| for | Solution must be constrained to individual torque limits for each motor, which are found from the electric motor lookup tables, motor speed, motor temperature, and DC bus voltage. These are instantaneous limits, meaning they change every time step. Note: Since each limit is unique for each motor, this is considered to be 4 separate constraints in the math. |
|  | Solution must be constrained to a maximum and minimum power limit. This power limit is the power draw at the accumulator, so the solution’s power draw is found from each motor’s torque output and efficiency at that operating condition, motor speed, as well as battery and inverter efficiencies. The last two may be implemented as a lookup table, or less accurately, approximated as constant. Maximum and minimum power limits are set by the driver. |
|  | The solution must be constrained to a maximum and minimum torque limit. These are set by the driver. This is mainly used in practice to limit acceleration, as power limiting usually only limits the top end. |
| for | The solution must be constrained to maximum and minimum slip angles. This is done to avoid selecting a high slip angle (~ > 0.3) as a solution to reducing tire Fx. This ultimately limits the amount of wheel slip allowed. These limits probably will be programmed constants. |

### Yaw Moment Objective Function

where

and

and

+ + +

+ + +

+ + +

+ + +

Note that the self-aligning moment contributions are left out of the yaw moment equation. This is because the left and right-side moments mostly cancel each other out, and the net contribution is significantly less than what the Fy and Fx tire forces contribute. This makes the algorithm less complex, because tire Mz lookup tables do not need to be included and linearized in real-time. However, this can certainly be implemented in the future.

### Longitudinal Acceleration Objective Function (TODO)

### Slip Angle Objective Function (TODO)

## Linearization

The non-linear relationships are the tire forces and motor efficiencies .

A local linearization technique is used to simplify the solving of the objective function. This involves:

1. linearizing the system around the current operating point
2. solving for the new torque solution via a QP solver,
3. Solve for the new slip ratio operating points at the torque solution
4. compare error between the torque solution and the non-linear torque computed by the lookup tables at the new operating point.
5. Repeat optimization if the error is above a threshold, or if an optimization cycle limit has not yet been hit. If we need to re-optimize, we start at the operating point of the last torque solution.

This process may require multiple optimization cycles, but it allows a simple, linear system to be solved and reflects the non-linearity of the vehicle in the solution.

### Tire Force Linearization

For a given cycle of the torque vectoring algorithm, the tire slip angles and normal forces are measured and then assumed constant. This makes the tire forces one-dimensional non-linear curves. Then we fit a linear equation to the lookup table by interpolating the able to find the current operating points and computing the derivative:

Note that in these equations, the partial derivative is taken at the operating point. That is:

Since we know what the tire lookup tables are beforehand, we can compute the derivates offline and store these as lookup tables on the microcontroller to speed up computation.

It is important to note that our decision variables are torques, not slip angles. Therefore, the objective functions should be functions of the torque. With the linearized tire forces, we can re-arrange to solve for the tire forces as a function of torque:

=

Therefore:

The tire friction circle relationship between lateral and longitudinal forces is reflected in the non-linear lookup table data, which is accounted for when we perform multiple local linearization.

## Conversion into a Convex Quadratic Programming Problem

To solve the optimization problem, we need to implement a solver for the problem defined above. There are hierarchical methods, genetic algorithms, etc, however we will use a quadratic programming (QP) solver which solve linear optimization problems, and which can be implemented in an embedded environment without significant computational requirements. [OSQP](https://osqp.org/docs/index.html) is the QP solver we chose as it is open-source and provides C and MATLAB interfaces.

OSQP solves optimization problems in the form of:

|  |  |
| --- | --- |
|  | Is a vector of the decision variables of length n. They must be all real numbers. |
|  | P is a matrix defining the quadratic part of the objective function. The P matrix must be positive semidefinite (PSD). The matrix must also be , which means it must be a real n x n matric that is symmetric (). |
|  | q is a vector defining the linear part of the objective functions. It has length n and must all be real numbers. |
|  | A is a matrix representing each constraint. Must be m x n and all real numbers. Each row (m) represents one linear constraint. |
| for all | L is the lower bounds for each constraint |
| for all | U is the upper bounds for each constraint |

### CVXGEN Solver

In addition to OSQP, [CVXGEN](https://cvxgen.com/docs/index.html) is an online tool which can be used to generate solver code in C and MATLAB for convex optimization problems. CVXGEN was used first for its simplicity. The QP problem format defined by OSQP was used to generate the solver. This is because we can switch between the CVXGEN-generated solver and OSQP without reformulating the data.

The code used to generate the solver is:

dimensions  
  # aka size of arrays to represent problem with  
  n = 4 # number of decision variables  
  m = 4 # number of constraints  
end  
  
parameters  
  # placeholders for problem data to be filled in with the generated solver  
  P(n, n) *symmetric psd*# matrix which defines the quadratic part of the objective functions  
  q(n) # vector defining the linear part of the objective functions  
  A(m, n) # matrix representing the linear constraints. Each row represents one constraint  
  l(m, 1) # column vector representing the lower bound for each constraint  
  u(m, 1) # column vector representing the upper bound for each constraint  
end  
  
variables  
  x(n) # vector of decision variables  
    
end  
  
minimize  
  (1/2) \* quad(x, P) + q'\*x # typical QP form, also identical to OSQP if we want to switch / compare solvers  
subject to  
  l **<=**A\*x **<=**u  
end

### Converting Objective Functions into P and q matrices

#### Yaw Moment Objective

With

And Mz being linear, this objective becomes quadratic as a function of the decision variables . We expand this:

With being a constant, we need to re-write in matrix form. Because our decision variables are wheel torques, we re-write the prior yaw moment equations:

+ + +

+ + +

+ + +

+ + +

Now we need to expand the moment equation and put it in matrix form, for easy use with our solver:

In matrix form the moment is , where is the vector of wheel torques, the matrix of coefficients multiplied by the wheel torques, and being the sum of the constants:

It should be noted, that when we linearize the tire forces, part of the Fy terms become linear functions of the torques, and therefore should be put into the matrix:

Re-arranged this gets us

Expanding above we get:

Moving all the terms to the matrix, we get:

Note, this is still a 1x4 matrix. The constant term is now:

Which is our final form for . This makes our objective function :

If we re-arrange this into the form (typical QP form), we get:

Next, we expand that to get the quadratic and linear parts of the first objective function:

is a quadratic term contributing to the matrix P. is the linear term contributing to the matrix q. is a constant and could be dropped because it does not change the value of the optimized decision variables.

#### Overall objective function

Since all the objectives take the same form:

In the cost function:

The quadratic terms of the cost function are , where

It should be noted that was found earlier to be 1x4, where n = 4 (number of decision variables). is 4x1, and so the matrix P becomes 4 x 4 (n x n), which is required by the QP solver. Now we can arrange it in the form that the QP solver wants:

It should also be noted that the P matrix must be positive semi-definite. This is guaranteed by ensuring all the objective functions take the form . Implementation of this part is simply a matter of passing the P matrix values to the QP solver.

#### Constraints

The constraints are quite straightforward, as it is just a matrix form of the linear constraint equations. For the last constraint, minimizing slip ratios, we have:

for

Solving in terms of the torque decision variables we get

In matrix form we get

However, the QP solver form does not support the constant matrix b. Instead, we include the constants in the limit vectors l and u, and convert this single constraint into 4 constraints with unique limits for each tire:

# Lookup Table Data Reduction

To implement the torque optimization on to an embedded system, the following lookup tables are needed:

Straight from MATLAB:

|  |  |  |  |
| --- | --- | --- | --- |
| Lookup Table Matrix | Breakpoint Arrays | Size & units | Max / Min Value |
| Tire force | * **Slip angle** 105] double [deg], 0.25 deg increments * **Slip ratio** 31] double [unitless], 0.02 increments * **Normal Force** 65] double [N], 25 N increments | 31x105x65 double  826.4 kB  Must be signed | 3,251.9 / -3,614.7 |
| Tire force | * **Slip angle** 105] double [deg], 0.25 deg increments * **Slip ratio** 31] double [unitless], 0.02 increments * **Normal Force** 65] double [N], 25 N increments | 31x105x65 double  826.4 kB  Must be signed | 3,742.6 / -3,635.7 |
| Electric motor Power Loss | * **Motor speed** [201] double [RPM], 100 RPM increments * **Motor current** [21] double [A], 5.25 A increments * **Motor temp** [3] double, 20 deg C increments | 201x21x3 double  49.5 kB  Can be unsigned | 6,361.6 / 0 |

With 1 Mb of flash, the lookup table data size needs to be significantly reduced. To do this we:

* Truncate values to integers because that precision is not needed. Therefore no need to scale the values to store in an int data type
* Reduce the number of breakpoints. This is okay because we can interpolate the data
* Use a smaller data type like int16\_t:

|  |  |  |  |
| --- | --- | --- | --- |
| Reduced Size Lookup Table | Breakpoint Arrays | Size & units | Max / Min Value |
| Tire force | * **Slip angle** 53] double [deg], 0.5 deg increments * **Slip ratio** 31] double [unitless], 0.02 increments * **Normal Force** 33] double [N], 50 N increments | 31x53x33 int16\_t  105.9 kB  Must be signed | 3,251 / -3,614 |
| Tire force | * **Slip angle** 53] double [deg], 0.5 deg increments * **Slip ratio** 31] double [unitless], 0.02 increments * **Normal Force** 33] double [N], 50 N increments | 31x53x33 int16\_t  105.9 kB  Must be signed | 3,742 / -3,635 |
| Electric motor Power Loss | * **Motor speed** [201] double [RPM], 100 RPM increments * **Motor current** [21] double [A], 5.25 A increments * **Motor temp** [3] double, 20 deg C increments | 201x21x3 int16\_t  24.7 kB  Can be unsigned | 6,361 / 0 |

The MATLAB /tire\_model/generate\_VCU\_tire\_data.m was used to reduce the MATLAB tire model and create a C file with the data declared in flash memory.