YASKAWA

MOTOMAN ROBOTICS

Motoplus-ROS Incremental Motion interface

Engineering Design Specifications

DOCUMENT NO:	M2092-EDS
DOCUMENT VER.:	2.0.0
DATE:	10/18/2023

Disclaimers

The information contained in this document is the proprietary and exclusive property of Yaskawa Motoman Robotics except as otherwise indicated. No part of this document, in whole or in part, may be reproduced, stored, transmitted, or used for design purposes without the prior written permission of Yaskawa Motoman Robotics.

The information contained in this document is subject to change without notice.

The information in this document is provided for informational purposes only. Yaskawa Motoman Robotics specifically disclaims all warranties, express or limited, including, but not limited, to the implied warranties of merchantability and fitness for a particular purpose, except as provided for in a separate software license agreement.

Privacy Information

This document may contain information of a sensitive nature. This information should not be given to persons other than those who are involved in the **Spatial Vision** project or who will become involved during the lifecycle

History

Revisions and Reviews			
Version	Person(s)	Description	Date
1.0.0	Tom Moolayil	Original version	11/01/2011
2.0.0	Eric Marcil	Updated functionality added over time	10/18/2023

Document Approval

Motoman:

Yaskawa America, Inc.
Motoman Robotics Division
100 Automation Way
Miamisburg, OH 45342
937-847-6200

Customer:

Company Name
Address
City, State, Zip
Phone

Approvals:

#	Name	Title	Organization/Dept.
1			
2			
3			
4			
5			
6			

#	Signature	Date
1		
2		
3		
4		
5		
6		

Table of Contents

Index of Figures

Index of Table

1 Overview

The ROS-Industrial program, initiated by Southwest Research Institute (SwRI), enables new applications and reduces project costs for industrial robotics. ROS-Industrial leverages the advanced capabilities of the Robot Operating System (ROS) software for powerful new industrial applications. This platform is usually used to calculate possible robot IK solutions by creating a virtual world identical to that of the real robot and using the obstacle/work space information to plan an optimal path to perform a task.

ROS industrial calculates a path and streams the way points to the MotoRos, MotoPlus application) running on the Yaskawa controller. The MotoRos application creates pulse increments that are sent to the controller command position to move the robot along the received path.

1.1 Scope

Yaskawa electric has released a new function that allows sending incremental motion at high rate. This new function should allow moving the robot without speed limitation. The implementation will require that:

1. The creation of a Motoplus application running on the controller (DX100, DX200, FS100, YRC1000 or YRC1000micro) to receive raw streaming data from the ROS side and use these streaming way points to execute the trajectory using the mpMeiIncrementMove function.
2. The ROS side needs to be changed to accommodate speed data along with the way-points. The current message type used in ROS comprises only of the trajectory points.

1.2 Objectives

The overall objectives of this project are:

1. To enable the robot to execute externally generated trajectories at full speed and smoothing as is appropriate during the course of executing any trajectory.
2. To create a Motoplus application that used the mpMeiIncrementMove or mpExRcsIncrementMove function in the algorithm .
3. To change the communication interface of ROS to incorporate velocity data along with way-points.
4. To implement restrictions on the incoming data from the PC to enforce safety and prevent damage to the robot.

2 Specifications

2.1 Architecture

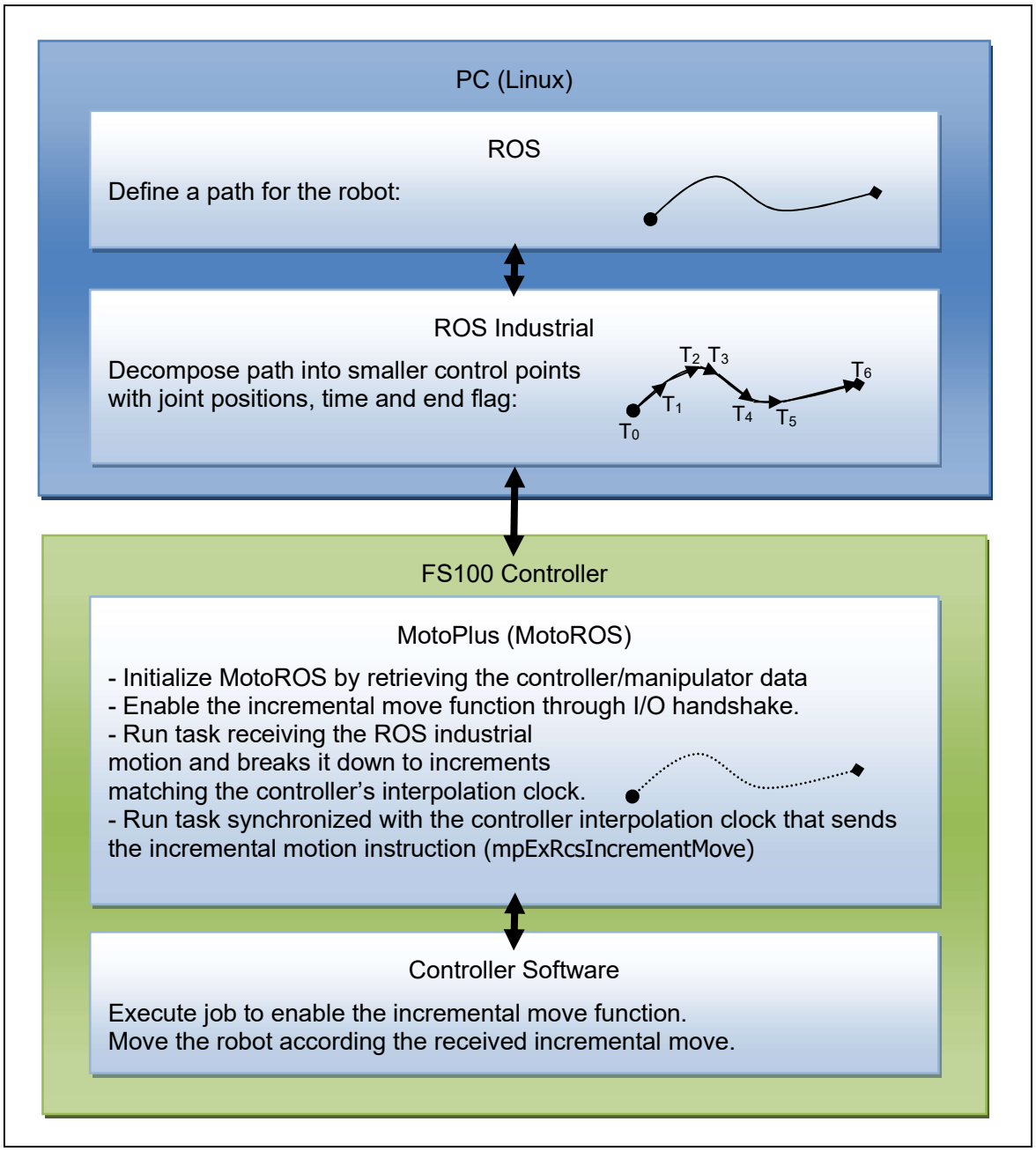

Figure 1: System Architecture

2.1.1 ROS to Motoplus

ROS-Industrial is responsible for generating the way-points and sending them to the robot controller. ROS will internally generate the way points and velocity information and send it via TCP/IP to the controller to interpret and use them as it sees fit.

2.1.2 Motoplus to ROS

Once Motoplus receives a point, it sends an acknowledgment to the ROS side to let it know it has received the way point and it is ready to receive subsequent points. This is the main communication that will happen between Motoplus to ROS.

Other communication will be to report the state of the system, current position, and extra services (such as reading or writing specific I/O, setting the active tool…)

2.1.3 INFORM to Motoplus

There is an inform job, INIT_ROS, which must be running on the controller to enable it to receive motion commands from the Motoplus application. It doesn't need to have any motion commands. The mpExRcsIncrementMove command only works when the output #889 is ON and the cursor is on a WAIT command. The INFORM job will look as follows:

```
NOP
DOUT OT#(890) OFF
DOUT OT#(889) OFF
TIMER T=0.05
DOUT OT#(889) ON
WAIT OT#(890)=ON
DOUT OT#(890) OFF
END
```

2.2 Communication Sequence/Flow

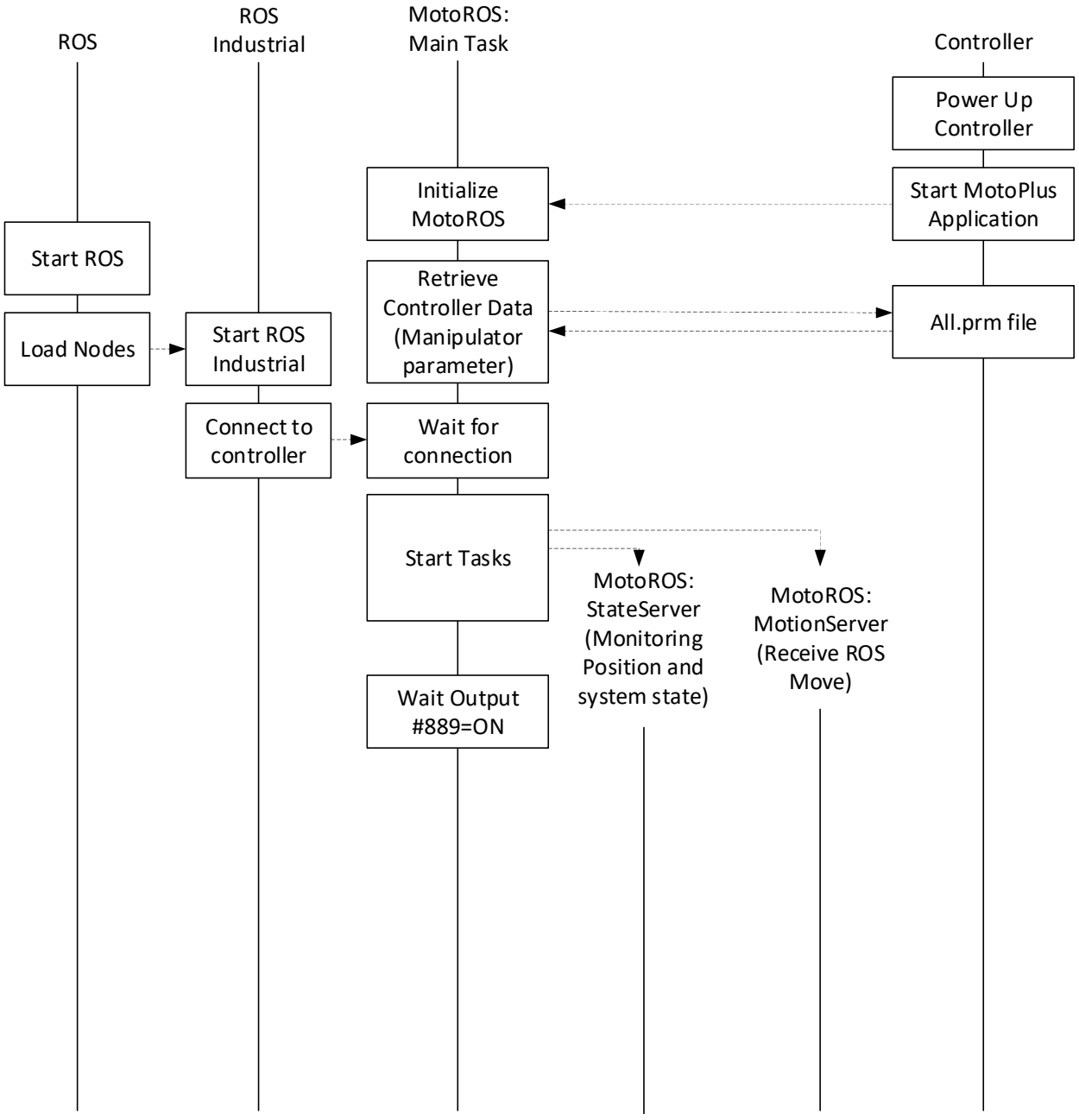

Figure 2: Start-up Sequence

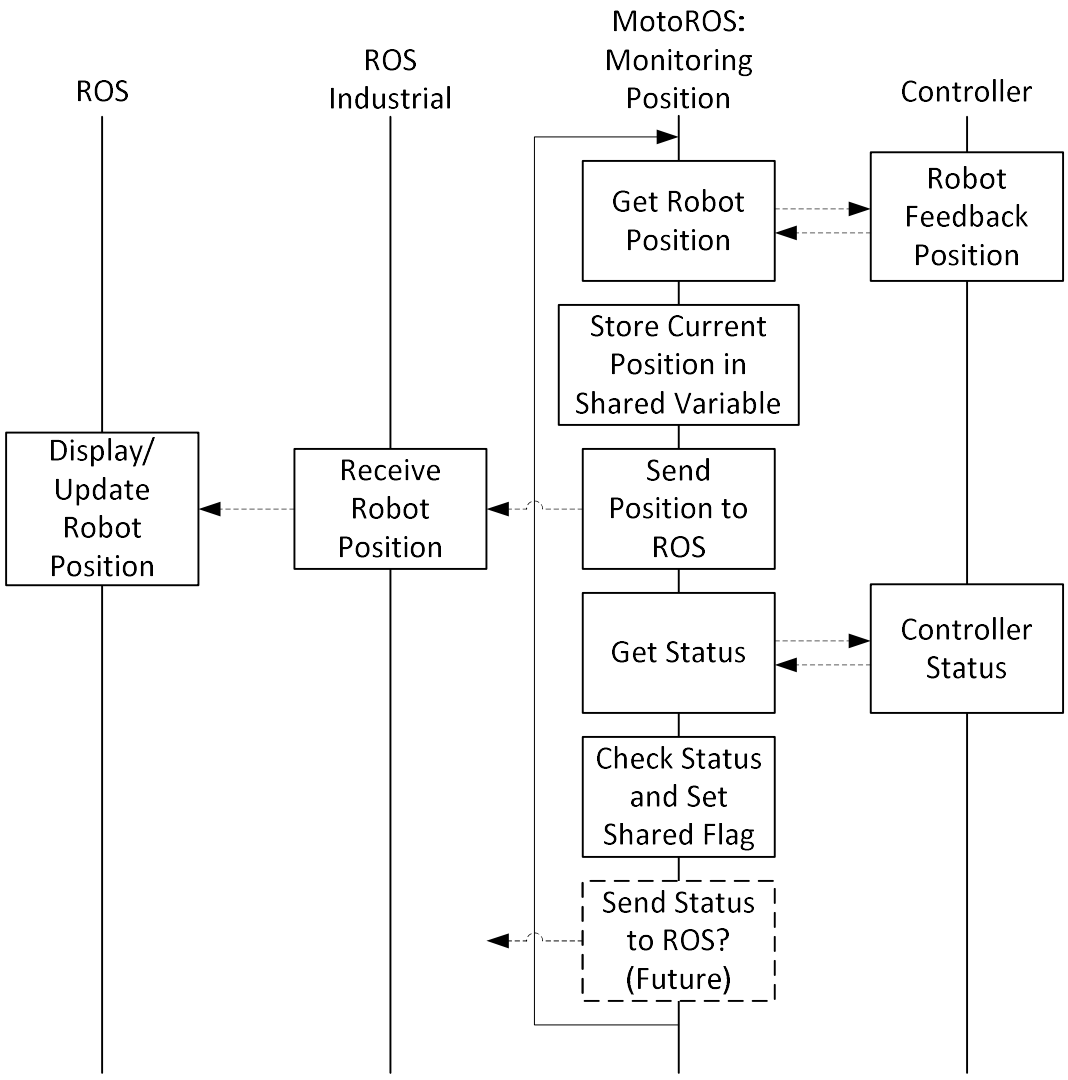

Figure 3: Monitoring Task Sequence

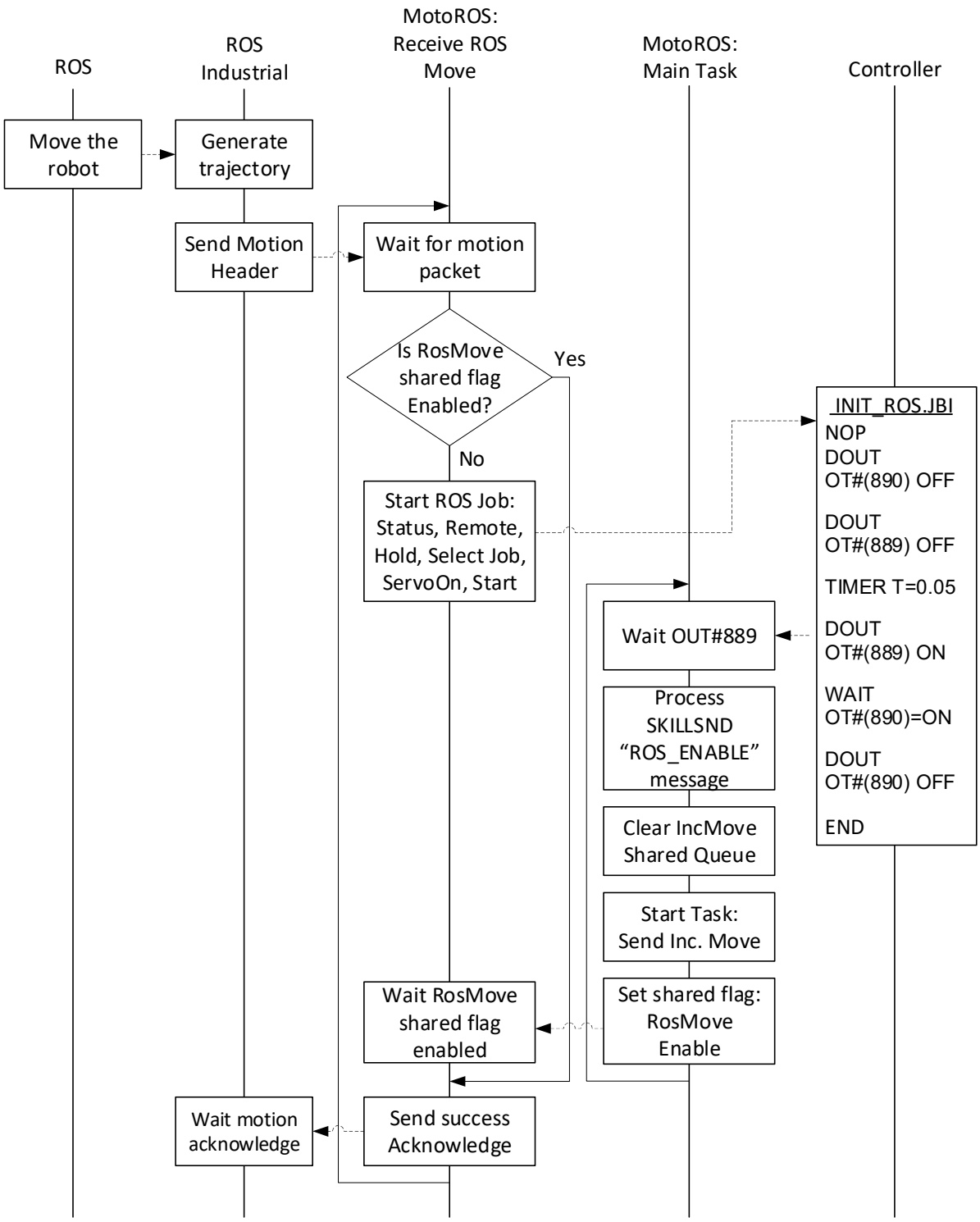

Figure 4: Receive ROS Move – Initialization Sequence

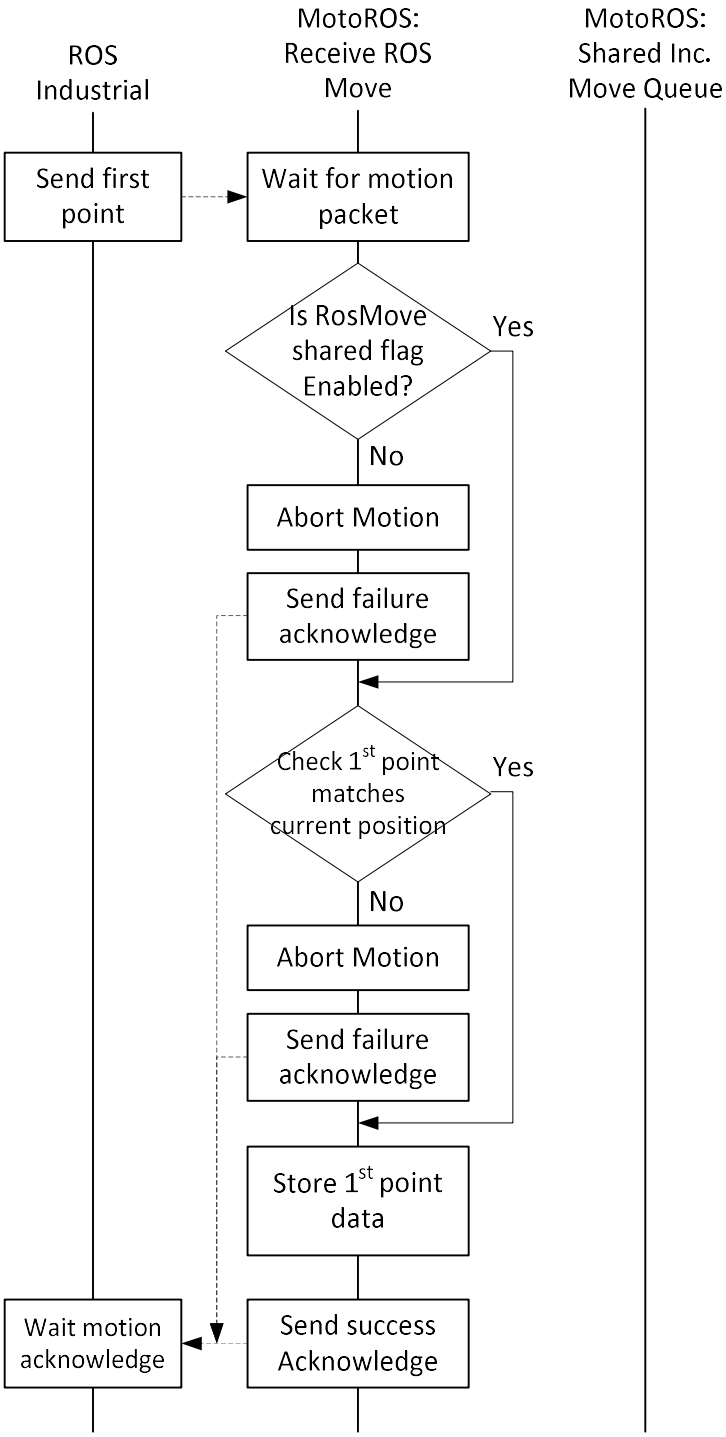

Figure 5: Receive ROS Move – First Point Sequence

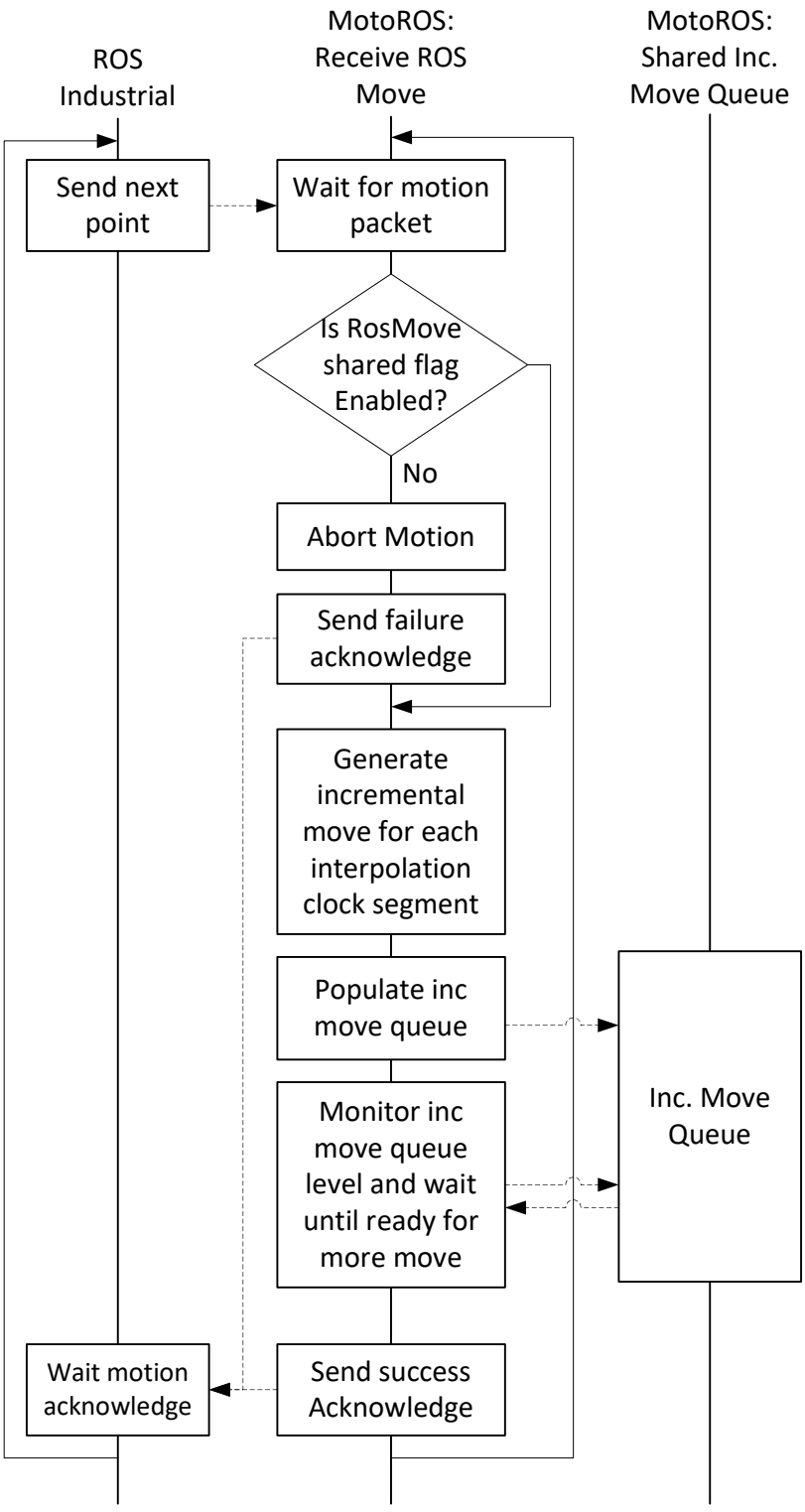

Figure 6: Receive ROS Move – Next Point Sequence

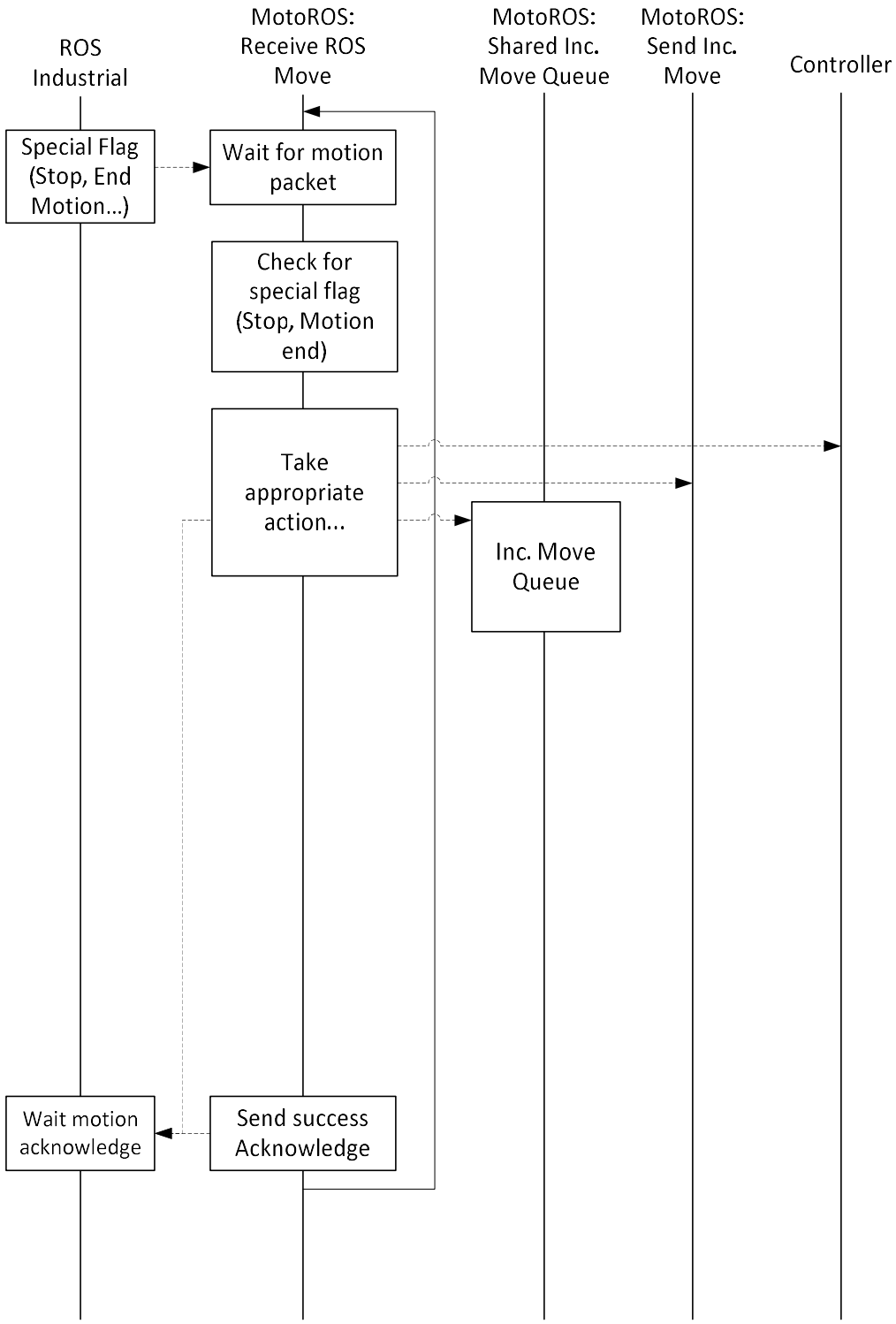

Figure 7: Receive ROS Move – Other Sequence

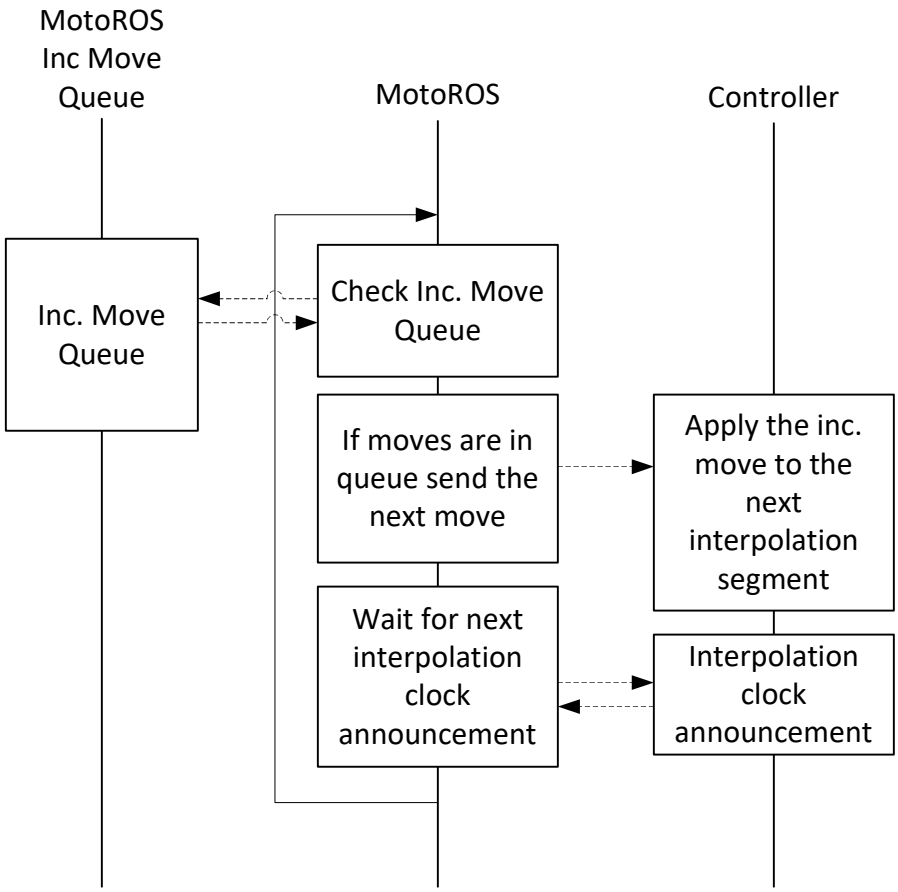

Figure 8: Send Incremental Move Task Sequence

2.3 Message and Data

2.3.1 Controller Class Data

In the initialization phase of the MotoROS library, the controller class is instantiated and caches all the values from the parameter extraction library for later use within the increment move function. Following are the functions (along with dummy input variables) that are called currently:

GP_isPflEnabled()

GP_getInterpolationPeriod(): Get the interpolation period in milliseconds.

GP_getNumberOfGroups(): Retrieves the Number of Defined Groups

For each group:

GP_getNumberOfAxes (ctrl_grp): Retrieves the number of axes of the group.

GP_getAxisMotionType: Gets the motion type of each axis in the group

*GP_getPulseToRad(int ctrlGrp, PULSE_TO_RAD *PulseToRad): Retrieves the pulse to radian conversion factors for each axis.*

GP_getAxisMotionType(int ctrlGrp, AXIS_MOTION_TYPE axisType): Gets the Pulse to meter conversion factors*

*GP_getFBPulseCorrection(int ctrlGrp, FB_PULSE_CORRECTION_DATA *correctionData): Retrieves the feedback pulse corrections for coupled motor axes.*

*GP_getMaxIncPerIpCycle(int ctrlGrp, int interpolationPeriodInMilliseconds, MAX_INCREMENT_INFO *mip): Get the maximum pulse increment per interpolation cycle.*

GP_getFeedbackSpeedMRegisterAddresses(int ctrlGrp, BOOL bActivateIfNotEnabled, BOOL bForceRebootAfterActivation, JOINT_FEEDBACK_SPEED_ADDRESSES registerAddresses): Obtains the MRegister CIO addresses that contain the feedback speed for each axis. Optionally enables this feature if not already enabled.*

GP_isBaxisSlave(int ctrlGrp, BOOL bBaxisIsSlave): Determines if B axis is automatically moved relative to other axes.*

GP_isSdaRobot(BOOL bIsSda): Determines if the robot is a dual-arm SDA.*

GP_isSharedBaseAxis(BOOL bIsSharedBaseAxis): Determines if the robot is an SDA that has a base axis which is shared over multiple control groups.*

Below is an example of the list of parameters which will be cached when the controller cache is initialized:

S_PULSE_TO_RAD = 82239.523438
L_PULSE_TO_RAD = 74502.703125
U_PULSE_TO_RAD = 78879.734375
R_PULSE_TO_RAD = 32594.931641
B_PULSE_TO_RAD = 47206.453125
T_PULSE_TO_RAD = 24382.703125
E_PULSE_TO_RAD = 0.000000

Interpolation cycle millisec = 4

S MaxInc = 1263
L MaxInc = 1040
U MaxInc = 1211
R MaxInc = 932
B MaxInc = 1351
T MaxInc = 1038

MAIN Percentage of maximum speed = 0.500000

2.3.2 ROS to MotoPlus

2.3.2.1 Message type 1: JOINT_TRAJ_HEADER

Details

2.3.2.2 Message type 14: JOINT_TRAJ_PT

The message called JOINT_TRAJ_PT type which includes position, velocity, acceleration, and time is designated as type 14. Once the Motoplus side received this message, it executed the callback for this type.

This contains the following information:

- sequence: TYPE: industrial::shared_types::shared_int (4 bytes)

  Value of -2 indicates START_TRAJECTORY_STREAMING which tells the MotoPlus side that ROS is going to start sending points

  Value of -3 indicates END_TRAJECTORY implying last point,

  Value of -4 indicates STOP_TRAJECTORY to stop motion

  Values of 0 or greater indicate actual sequence or the incoming point (0 indicates start point, 1 indicates the first point, 2 indicates second etc.)

- positions: TYPE: industrial::joint_data (40 bytes): contains joint configuration in radians

- velocities: TYPE: industrial::joint_data (40 bytes): contains velocity in radians/sec

- accelerations: TYPE: industrial::joint_data (40 bytes): contains acceleration in radians/sec/sec

- time_from_start: TYPE: industrial::shared_types::shared_real (4 bytes) : float containing time in seconds

The industrial::joint_data type is of size 40bytes and has a member “joints” which is an array that will contain 8 points (8 is the maximum number of axes) of 4 bytes each (8x4 = 32bytes) (There are delimiters after each point of size 1byte(8x1 = 8bytes)) . The underlying type of all of the above is “float”.

2.3.3 MotoPlus to ROS

2.3.3.1 Acknowledge message

Motoplus sends an acknowledgement each time it receives a point indicating it has received the point and is ready for the next. This is mainly for timing and coordination purposes.

Each time a new point comes in the motion interface is triggered and the joint information is moved to a buffer(using an addPoint function). Once the point is moved to a buffer and the motion has been executed Motoplus sends back a “joint acknowledgment” to the ROS side after receiving which ROS sends in the next point.

The joint acknowledgement type will be in the following formats:

ReplyTypes::SUCCESS – Received point and executed!!

ReplyTypes::FAILURE – Motoplus doesn’t recognize the incoming message type.

ReplyTypes::INVALID – The data is not in the correct format e.g point has 5 joint values but the robot is 6 axes.

2.3.4 Increment Move Queue

2.3.5 MotoPlus to Controller

The MotoPlus application will send the incremental move using the function:
mpExRcsIncrementMove(MP_POS_DATA *src_p)

```
Typedef struct {
CTRLG_T ctrl_grp;          // control group
CTRLG_T m_ctrl_grp;        // master control group (coordinated motion)
```

```
CTRLG_T s_ctrl_grp;              // slave control group (coordinated motion)
MP_GRP_POS_INFO grp_pos_info[MP_GRP_NUM];  // see structure detail
} MP_POS_DATA;
```

For a single arm system R1:

```
ctrl_grp = 1
m_ctrl_grp = 0
s_ctrl_grp = 0
```

```
Typedef struct {
MP_POS_TAG pos_tag;                  // see structure detail
long pos[MP_GRP_AXES_NUM];           // position information
} MP_GRP_POS_INFO

Typedef struct {
UCHAR data[8];                       // defines the axes, tool and user frame used
CTRLG_T ei_ctrl_grp;                 // used only with EIMOV to set the control group
} MP_POS_TAG
```

MP_POS_TAG defines the axes, tool and user frame used.
MP_POS_TAG.data[0]: is for the axis used. This is a bitwise value with the first bit corresponding to the first axis and the 8th bit to the 8th axis. So for:

- 6-axis robot: pos_tag.data[0]=63
- 7-axis robot: pos_tag.data[0]=127.

MP_POS_TAG.data[2]: defines the tool file number
MP_POS_TAG.data[3]: defines the coordinate system.

- In our case, it should be set to MP_INC_PULSE_DTYPE

MP_POS_TAG.data[4]: defines the user frame number

MP_GRP_POS_INFO.pos[MP_GRP_AXES_NUM] define the pulse increment for each axis.

2.4 Interpolation of Pulse Increment

The MotoROS will receive trajectory points from the ROS Industrial. The trajectory points will include sequence number, time stamp, position (absolute), velocity and acceleration for each joint using angular radians units. The points maybe spread along a path at different spacing or time interval. The MotoRos application will need to interpolate the path between those points and determine the corresponding incremental move to send to the controller at the controller set interpolation cycle. The following are the calculation to be implemented for this interpolation.

2.4.1 Constant for a specific controller

InterpolationCycle: Controller interpolation cycle time. (Usually 4 ms)

NbAxis: Number of axes (joints) on the manipulator.

RadToPulse[NbAxis]: Conversion ratio from radian to pulses for each axis.

MaxSpeedPulse[NbAxis]: Maximum rated speed in pulses/InterpolationCycle for each axis.

MaxAccelPulse[NbAxis]: Maximum rated acceleration in pulses/InterpolationCycle^2 for each axis.

2.4.2 Variables

nextInterCycleInc: The amount of time that should be added to the calculation time

calculationTime: Time elapse since the beginning of the motion

prevRosPoint[NbAxis]: Previous ROS control point (includes position, velocity, acceleration, time)

newRosPoint[NbAxis]: New ROS control point (includes position, velocity, acceleration, time)

prevInterPoint[NbAxis]: Previous interpolated position in pulses (includes position and time).

newInterPoint[NbAxis]: New interpolated position in pulses (includes position and time).

prevIncPulse[NbAxis]: New pulse increment for the next interpolation cycle (includes increment pulse, time increment, time)

newIncPulse[NbAxis]: New pulse increment for the next interpolation cycle (includes increment pulse, time increment, time)

interAccel[NbAxis]: Linear interpolation of acceleration at a given time.

2.4.3 Algorithm

For each new ROS point:

Calculate acceleration ratio

While time is smaller than new ROS point time

Increment calculation time by next interpolation cycle

If next interpolation cycle is smaller than the controller interpolation cycle, make it equal.

If calculation time is smaller than new ROS point time

Set new time to calculation time

For each axis

Calculate new acceleration for the current calculation time

Calculate new velocity for the current calculation time

Calculate new position for the current calculation time

Else (if calculation time is equal or larger than new ROS point time)

If calculation time is larger than new ROS point time

Set the next interpolation increment to the different between the two

Set the calculation time equal to the new ROS point time

Set new time to new ROS point time

For each axis

Set new acceleration to new ROS point acceleration

Set new velocity to new ROS point velocity

Set new position to new ROS point position

Convert new position in pulses

Calculate new pulse increment by subtracting previous pulse position from new pulse position.

Check speed and acceleration limits

Set new pulse increment to queue

2.4.4 Calculation

Acceleration ratio:

```
AccelRatio[i] = (newRosPoint.accel[i] - prevRosPoint.accel[i])
                / (newRosPoint.time - prevRosPoint.time)
```

New time:

```
newInterPoint.time = calculationTime
```

New time increment:

```
newInterPoint.IncTime = newInterPoint.time - prevInterPoint.time
```

New acceleration:

```
newInterPoint.accel[i] = prevRosPoint.accel[i]
                         + AccelRatio[i] * (newInterPoint.time - prevRosPoint.Time)
```

New velocity:

```
newInterPoint.speed[i] = prevInterPoint.speed[i]
                         + prevRosPoint.accel[i] * newInterPoint.IncTime
```

New position:

```
newInterPoint.speed[i] = prevInterPoint.position[i]
                         + prevRosPoint.speed[i] * newInterPoint.IncTime
                         + prevRosPoint.accel[i] * newInterPoint.IncTime^2 / 2
```

2.4.5 Check speed and acceleration

Check for each axis:

Speed:

MaxSpeedPulse[i] =

If abs(newIncPulse[i]) > MaxSpeedPulse generate error

Acceleration:

MaxAccelPulse[i] =

If abs(newIncPulse[i] - prevIncPulse[i]) > MaxAccelPulse generate error

2.5 FSU Speed Limit handling

Note: This is currently under development under the PR#542 on Github.

2.5.1 Overview

The FSU (Functional Safety Unit) works independently from the controller, therefore all FSU function will work properly whether MotoRos is used or not. The issue is that when activated, there is no explicit feedback to ROS. In the case that alarms are generated by the FSU, the ROS side will be notified of the alarm state like any other alarm. The alarm will need to be reset, and motion reinitialized from the current position.

The issue is when function like Speed Limit is enabled, any motion sent by ROS at speed higher than the speed limit will be reduce, this will affect the robot motion and the robot will not reach it's expected end position. The problem is that there is no feedback available at this time from the FSU to indicate that the motion was dynamically reduced that can be used to relay the information to the ROS side.

To work around this issue, the MotoRos driver checks if the incremental pulses sent on the previous iteration matches the change in the robot command position. If it doesn't it means that the FSU Speed Limit is actively limiting the speed and rejecting some of the command.

By keeping track of the previous iteration values, the amount of increment processed is calculated and the unprocessed part is resent. When there are unprocessed pulses, the reading of the incremental queue is limited to one and then skip further reading until the previous increment is completely processed.

The speed associated with an increment is also tracked so that if the speed limit is removed, the unprocessed pulses won't be sent all at once and exceed the commanded speed.

2.5.2 Calculations

The FSU Speed Limit handling is done in the Ros_MotionServer_IncMoveLoopStart at every iteration cycle.

Motion Processing Condition

Originally when the increment queue was empty, the processing would be skipped motion processing would be skipped, but when the FSU Speed Limit is enabled, pulse increments from the previous iteration might not have been accepted by the controller and need to be resent. So, the *hasUnprocessedData* flag was added to the condition to continue motion processing until all data is processed even if the increment queue is empty.

Reading the Increment Queue

During the reading of the increment queue, we also added a flag, *skipReadingQ*, to skip

reading more data from the increment queue. Otherwise, when the FSU Speed Limit is enabled, the unprocessed data would just keep in accumulate into a large value, gradually merging all the data together and the motion path would no longer be matching the intended path sent by ROS. So, whenever the amount of unprocessed pulses is larger than the maximum amount of pulses that can be on the next iteration (*MaxSpeed*), the *skipReadingQ* is set to true.

FSU Speed Limit Check

Note that processing is always considering all the axes of a group together. So, for a motion increment to be considered processed, the motion of all axes in that group must be complete. If any of the axes have unprocessed pulses, then the whole increment is considered unprocessed. There are several steps in this section.

- Record the motion speed sent by ROS as the *maxSpeed,* pulses per iteration (usually 4 ms). If there is a large number of unprocessed pulses and the FSU speed limit is suddenly disabled, if all the unprocessed data is sent at once, it might create a speed burst that would exceed the speed originally commanded by ROS. So, the *maxSpeed* is tracked and the driver will never send number of pulses exceeding this *maxSpeed*. The speed may also vary through a trajectory, so we also track for how many pulses should this maxSpeed be applied. This is stored in the associated variable *maxSpeedRemain*, which tracks for how many more pulses should that maxSpeed be applied.

- Check if pulses are missing from last iteration increment. To do this, the current controller command position is retrieved and the previous command position substract from it to get the *processedPulses* amount. If it doesn't match the amount of increment sent on the previous iteration, then some pulses are missing, and the number of unprocessed pulses needs to be added to this iteration *toProcessPulses*. If a new pulse increment was also read from the queue on this iteration, it is also added to the *toProcessPulses* variable.

- If there are missing pulses, *isMissingPulse* flag, then the FSU speed limit is active, and the management of those unprocessed pulses is needed.

    - As mentioned before, to prevent going faster than ROS requested speed, we track the *maxSpeed* and *maxSpeedRemain*. This is done for a maximum of two queue increments at the time, the previous (*prev* prefix) and the current one. The *processedPulses* are removed from the *prevMaxSpeedRemain*. When the *prevMaxSpeedRemain* gets to zero, it means that all the pulses for that increment was processed, the current increment is then transferred to the previous increment variables and a new increment can be retrieved from the increment queue. Otherwise, if the *prevMaxSpeedRemain* is greater than zero, the *skipReadingQ* flag is set to true.

    - The number of pulses to be sent for the current iteration is then determined based on the minimum between the *toProcessPulses* variable and the *prevMaxSpeed*. The pulse value is then set to the *moveData* structure that will be sent to the controller in the following step.

Send pulse increment to the controller

The moveData structure is sent to the controller and the return value is retrieved. If non-zero, the return value is analyzed to determine the cause and set user feedback.

Clean-up

If the Motion Processing is not taking place for whatever reason (alarm, mode change…) the *Ros_MotionServer_IncMoveLoopStart* function internal variables are reset to make sure that once processing is resumed there are no remaining values from the previous motion.

2.5.3 Example

This is an example of the motion processing while the FSU Speed Limit is enabled.

	Requested Increments in Queue				
	Data 1	Data 2	Data 3	Data 4	Data 5
S	100	90	80	0	0
L	100	90	80	0	0
U	100	90	80	0	0
R	100	90	80	0	0
B	100	90	80	0	0
T	100	90	80	0	0

	FSU Speed Limits
S	40
L	40
U	50
R	50
B	60
T	60

	Cycle 1							
	Moved	Unprocessed				New		ToProcess	Sent
		Prev		Curr		From Data Queue			
		Remain	MaxSpeed	Remain	MaxSpeed	Remain	MaxSpeed		
S		0	0	0	0	100	100	100	100
L		0	0	0	0	100	100	100	100
U		0	0	0	0	100	100	100	100
R		0	0	0	0	100	100	100	100
B		0	0	0	0	100	100	100	100
T		0	0	0	0	100	100	100	100
		Get New	1						

	Cycle 2							
	Moved	Unprocessed				New		ToProcess	Sent
		Prev		Curr		From Data Queue			
		Remain	MaxSpeed	Remain	MaxSpeed	Remain	MaxSpeed		
S	40	0	0	60	100	90	90	150	100
L	40	0	0	60	100	90	90	150	100
U	50	0	0	50	100	90	90	140	100
R	50	0	0	50	100	90	90	140	100
B	60	0	0	40	100	90	90	130	100
T	60	0	0	40	100	90	90	130	100
		Get New	2						

	Cycle 3								
	Moved	Unprocessed				New		ToProcess	Sent
		Prev		Curr		From Data Queue			
		Remain	MaxSpeed	Remain	MaxSpeed	Remain	MaxSpeed		
S	40	20	100	90	90	0	0	110	90
L	40	20	100	90	90	0	0	110	90
U	50	0	100	90	90	0	0	90	90
R	50	0	100	90	90	0	0	90	90
B	60	0	100	70	90	0	0	70	70
T	60	0	100	70	90	0	0	70	70
		Skip	2						

	Cycle 4								
	Moved	Unprocessed				New		ToProcess	Sent
		Prev		Curr		From Data Queue			
		Remain	MaxSpeed	Remain	MaxSpeed	Remain	MaxSpeed		
S	40	0	100	70	90	80	80	150	100
L	40	0	100	70	90	80	80	150	100
U	50	0	100	40	90	80	80	120	100
R	50	0	100	40	90	80	80	120	100
B	60	0	100	10	90	80	80	90	90
T	60	0	100	10	90	80	80	90	90
		Get New	3						

	Cycle 5								
	Moved	Unprocessed				New		ToProcess	Sent
		Prev		Curr		From Data Queue			
		Remain	MaxSpeed	Remain	MaxSpeed	Remain	MaxSpeed		
S	40	30	90	80	80	0	0	110	80
L	40	30	90	80	80	0	0	110	80
U	50	0	90	70	80	0	0	70	70
R	50	0	90	70	80	0	0	70	70
B	60	0	90	30	80	0	0	30	30
T	60	0	90	30	80	0	0	30	30
		Skip	3						

	Cycle 6								
	Moved	Unprocessed				New		ToProcess	Sent
		Prev		Curr		From Data Queue			
		Remain	MaxSpeed	Remain	MaxSpeed	Remain	MaxSpeed		
S	40	0	90	70	80	0	0	70	70
L	40	0	90	70	80	0	0	70	70
U	50	0	90	20	80	0	0	20	20
R	50	0	90	20	80	0	0	20	20
B	30	0	90	0	80	0	0	0	0
T	30	0	90	0	80	0	0	0	0
		Get New	4						

	Cycle 7								
	Moved	Unprocessed				New		ToProcess	Sent
		Prev (Data1)		Curr (Data2)		Data3			
		Remain	MaxSpeed	Remain	MaxSpeed	Remain	MaxSpeed		
S	40	30	80	0	0	0	0	30	30
L	40	30	80	0	0	0	0	30	30
U	20	0	80	0	0	0	0	0	0
R	20	0	80	0	0	0	0	0	0
B	0	0	80	0	0	0	0	0	0
T	0	0	80	0	0	0	0	0	0
		Skip	4						

	Cycle 8								
	Moved	Unprocessed				New		ToProcess	Sent
		Prev (Data1)		Curr (Data2)		Data3			
		Remain	MaxSpeed	Remain	MaxSpeed	Remain	MaxSpeed		
S	30	0	80	0	0	0	0	0	0
L	30	0	80	0	0	0	0	0	0
U	0	0	80	0	0	0	0	0	0
R	0	0	80	0	0	0	0	0	0
B	0	0	80	0	0	0	0	0	0
T	0	0	80	0	0	0	0	0	0
		Get New	5						