$signals string variables

Robot as server

.PROGRAM init\_kawa()

;Initiate driver

ONE shutdown\_kawa ;Calls program when error occurs

keep\_active\_signal = 2010 ; Define signal number

signal keep\_active\_signal ;Set signal to TRUE

;Initiate movement pose queue

queue\_size = 50

queue\_front = 0

queue\_rear = 0

;Initiate tcp string buffer

$tcp\_buffer = ""

seq\_num\_old = 0

.END

.PROGRAM recv\_tcp\_serv () ;

;

$recv\_tcp\_serv\_state = "0, initiating"

.recv\_port = 9814

.timeout\_recv = 0.01

.num = 0

.ret\_listen = -1

;.recv\_sockid = -1

$recv\_tcp\_serv\_state = "Connecting on port " + $ENCODE(.recv\_port)

;

DO

TCP\_LISTEN .ret\_listen, .recv\_port ;creates socket and waits for connection request

IF .ret\_listen < 0 THEN

TWAIT 0.1

END

UNTIL .ret\_listen >= 0

;

DO

TCP\_ACCEPT .recv\_sockid, .recv\_port, 1 ;accepts socket

UNTIL .recv\_sockid > 0

;

$recv\_tcp\_serv\_state = "Connected and can start receiving poses through sockid: " + $ENCODE(.recv\_sockid)

;

WHILE sig(keep\_active\_signal) DO

.num = 0

tcp\_recv .ret, .recv\_sockid, .$tcp\_message[1], .num, .timeout\_recv, 1

;$recv\_tcp\_serv\_state = "Current state tcp comms: " + $ERROR(.ret)

IF .ret >= 0 THEN ;If we received stuff, add message to internal tcp\_buffer

IF .num > 0 THEN

for .i = 1 to .num

$tcp\_buffer = $tcp\_buffer + .$tcp\_message[.i]

END

END

ELSE

TWAIT 0.01

END

END

$recv\_tcp\_serv\_state = "Closing connection"

TCP\_CLOSE .ret, .recv\_sockid

IF .ret < 0 THEN

$recv\_tcp\_serv\_state = "Closing connection error: " + $ERROR(.ret)

TCP\_CLOSE .ret, .recv\_sockid

END

TCP\_END\_LISTEN .ret, .recv\_port

$recv\_tcp\_serv\_state = "Closed succesfully"

.END

.PROGRAM send\_pos\_serv()

;

$send\_server\_state = "Initiating"

.send\_port = 12014

.send\_msgs\_per\_sec = 10

.timeout\_recv = 0.1

.num = 0

.ret\_listen = -1

.ret\_sockid = -1

;

$send\_server\_state = "Listening on port " + $ENCODE(.send\_port)

;

DO

TCP\_LISTEN .ret\_listen, .send\_port

IF .ret\_listen < 0 THEN

TWAIT 0.1

END

UNTIL .ret\_listen >= 0

;

DO

TCP\_ACCEPT .send\_sockid, .send\_port, 1

UNTIL .send\_sockid > 0

;

$send\_server\_state = "Connected and can start sending current poses through sockid" + $ENCODE(.send\_sockid)

;

WHILE sig(keep\_active\_signal) DO

CALL encode\_pose(.$pose)

.$send\_buf[1] = .$pose

TCP\_SEND .ret, .send\_sockid, .$send\_buf[1], 1, .timeout\_recv

TWAIT (1 / .send\_msgs\_per\_sec)

END

;

$send\_server\_state = "Closing connection"

;

TCP\_CLOSE .ret, .send\_sockid

IF .ret < 0 THEN

$recv\_tcp\_serv\_state = "Closing connection error: " + $ERROR(.ret)

TCP\_CLOSE .ret, .send\_sockid

END

TCP\_END\_LISTEN .ret, .send\_port

;

.END

.PROGRAM read\_tcp\_buffer()

;

$read\_tcp\_buffer\_state = "Initiating"

;

.$start\_of\_msg0 = $CHR(2)

.$start\_of\_msg1 = $CHR(1)

.$end\_of\_msg = $CHR(3)

.$temp = "init"

;

start:

WHILE sig(keep\_active\_signal) DO

WAIT len($tcp\_buffer) > 0

;PRINT "Message: ", $msg

.$temp = $DECODE($tcp\_buffer, .$start\_of\_msg0, 0) ;Delete everything up to first start\_of\_message

IF len($tcp\_buffer) > 0 THEN ;Check if TCP buffer is not empty, can be empty of no start\_of\_message available.

.$temp = $DECODE($tcp\_buffer, "|", 0) ;Store characters upto seperator as variable

.$seperator = $DECODE($tcp\_buffer, "|", 1) ;Remove pipe seperator

IF .$temp == .$start\_of\_msg0 THEN ;Check if start\_of\_message was indeed found

WAIT len($tcp\_buffer) > 2

;PRINT "yes1!"

.$temp = $DECODE($tcp\_buffer, "|", 0) ;Store characters upto seperator as variable

.$seperator = $DECODE($tcp\_buffer, "|", 1) ;Remove pipe seperator

IF .$temp == .$start\_of\_msg1 THEN ;Check if start\_of\_heading was found

WAIT len($tcp\_buffer) > 3

;PRINT "yes2!"

.$temp = $DECODE($tcp\_buffer, "|", 0) ;Store characters upto seperator as variable

.msg\_length = VAL(.$temp)

WAIT len($tcp\_buffer) >= .msg\_length

;print "Message length: ", .msg\_length

.$full\_message = $DECODE($tcp\_buffer, .$end\_of\_msg, 0) ;store everything up to end\_of\_message

;print .$full\_message

IF len(.$full\_message) == (.msg\_length - 1) THEN

; print "YES!!!!!", .$full\_message

CALL decode\_traj\_pt(.$full\_message, .msg\_id, .prog\_speed, .prog\_acc, .prog\_accel, .prog\_decel, .prog\_break, .joint0, .joint1, .joint2, .joint3, .joint4, .joint5)

CALL ins\_rear\_queue(.msg\_id, .prog\_speed, .prog\_acc, .prog\_accel, .prog\_decel, .prog\_break, .joint0, .joint1, .joint2, .joint3, .joint4, .joint5)

ELSE

$read\_tcp\_buffer\_state = "error"

PRINT "oh no :( 3"

;call shutdown error!

END

ELSE

$read\_tcp\_buffer\_state = "error"

PRINT "oh no :( 2"

;call shutdown error!

END

ELSE

$read\_tcp\_buffer\_state = "error"

PRINT "oh no :( 1"

;call shutdown error!

END

ELSE

$read\_tcp\_buffer\_state = "error"

;print "TCP buffer empty!"

TWAIT 0.5

END

END

.END

.PROGRAM encode\_pose (.$pose) ;

HERE .#CP

DECOMPOSE .CP[0] = .#CP

.$S = "|"

.$pose = $CHR (2) + $ENCODE (/L, .$S, .CP[0], .$S, .CP[1], .$S, .CP[2], .$S, .CP[3], .$S, .CP[4], .$S, .CP[5], .$S) + $CHR (3) + $CHR (10)

.END

.PROGRAM decode\_traj\_pt (.$full\_message,.msg\_id,.prog\_speed,.prog\_acc,.prog\_accel,.prog\_decel,.prog\_break,.joint0,.joint1,.joint2,.joint3,.joint4,.joint5) ;

;

.$temp = $DECODE(.$full\_message, "|", 1) ; remove seperator

.$msg\_id = $DECODE(.$full\_message, "|", 0) ; read until next seperator

.msg\_id = VAL(.$msg\_id)

;

IF not .msg\_id == 11 and not .msg\_id == 12 THEN

$shutdown\_reason = "Trajectory decoding failed at ID"

CALL shutdown\_kawa

RETURN

END

;

.$temp = $DECODE(.$full\_message, "|", 1)

.$seq\_num = $DECODE(.$full\_message, "|", 0)

.seq\_num = VAL(.$seq\_num)

;

IF NOT .seq\_num == seq\_num\_old + 1 THEN

$shutdown\_reason = "Sequence number of received trajectory points mismatch. Expected:" + $ENCODE(seq\_num\_old) + " but got:" + $ENCODE(.seq\_num)

CALL shutdown\_kawa

RETURN

ELSE

seq\_num\_old = .seq\_num

END

;

.$temp = $DECODE(.$full\_message, "|", 1)

.$prog\_speed = $DECODE(.$full\_message, "|", 0)

.prog\_speed = VAL(.$prog\_speed)

;

.$temp = $DECODE(.$full\_message, "|", 1)

.$prog\_acc = $DECODE(.$full\_message, "|", 0)

.prog\_acc = VAL(.$prog\_acc)

;

.$temp = $DECODE(.$full\_message, "|", 1)

.$prog\_accel = $DECODE(.$full\_message, "|", 0)

.prog\_accel = VAL(.$prog\_accel)

;

.$temp = $DECODE(.$full\_message, "|", 1)

.$prog\_decel = $DECODE(.$full\_message, "|", 0)

.prog\_decel = VAL(.$prog\_decel)

;

.$temp = $DECODE(.$full\_message, "|", 1)

.$prog\_break = $DECODE(.$full\_message, "|", 0)

.prog\_break = VAL(.$prog\_break)

;

.$temp = $DECODE(.$full\_message, "|", 1)

.$joint0 = $DECODE(.$full\_message, "|", 0)

.joint0 = VAL(.$joint0)

;

.$temp = $DECODE(.$full\_message, "|", 1)

.$joint1 = $DECODE(.$full\_message, "|", 0)

.joint1 = VAL(.$joint1)

;

.$temp = $DECODE(.$full\_message, "|", 1)

.$joint2 = $DECODE(.$full\_message, "|", 0)

.joint2 = VAL(.$joint2)

;

.$temp = $DECODE(.$full\_message, "|", 1)

.$joint3 = $DECODE(.$full\_message, "|", 0)

.joint3 = VAL(.$joint3)

;

.$temp = $DECODE(.$full\_message, "|", 1)

.$joint4 = $DECODE(.$full\_message, "|", 0)

.joint4 = VAL(.$joint4)

;

.$temp = $DECODE(.$full\_message, "|", 1)

.$joint5 = $DECODE(.$full\_message, "|", 0)

.joint5 = VAL(.$joint5)

;

.END

.PROGRAM ins\_rear\_queue (.msg\_id,.prog\_speed,.prog\_acc,.prog\_accel,.prog\_decel,.prog\_break,.joint0,.joint1,.joint2,.joint3,.joint4,.joint5) ;

; GOTO command is necessary because AS language does not support ifelse statements

;

IF queue\_front == 1 AND queue\_rear == queue\_size -1 OR queue\_front == queue\_rear + 1 THEN ;Queue is full, shutdown

$shutdown\_reason = "Queue overload"

CALL shutdown\_kawa

RETURN

END

IF queue\_rear == 0 THEN ; Queue is empty, set front and rear pointers

queue\_rear = queue\_rear + 1

queue\_front = queue\_front + 1

GOTO add\_queue

END

IF queue\_rear == queue\_size - 1 AND queue\_front > 1 THEN ;if the rear queue pointer is at max size, loop to front

queue\_rear = 1

GOTO add\_queue

ELSE ; Move the rear one back

queue\_rear = queue\_rear + 1

GOTO add\_queue

END

;

add\_queue:

queue[queue\_rear, 0] = .msg\_id

queue[queue\_rear, 1] = .prog\_speed

queue[queue\_rear, 2] = .prog\_acc

queue[queue\_rear, 3] = .prog\_accel

queue[queue\_rear, 4] = .prog\_decel

queue[queue\_rear, 5] = .prog\_break

queue[queue\_rear, 6] = .joint0

queue[queue\_rear, 7] = .joint1

queue[queue\_rear, 8] = .joint2

queue[queue\_rear, 9] = .joint3

queue[queue\_rear, 10]= .joint4

queue[queue\_rear, 11] = .joint5

;

exit:

.END

.PROGRAM del\_front\_queue()

; GOTO command is necessary because AS language does not support ifelse statements

if queue\_front == 0 THEN ;Queue is empty, nothing to delete

GOTO exit

END

if queue\_front == queue\_rear AND queue\_front != 0 THEN ; Queue has one object, reset queue

queue\_front = 0

queue\_rear = 0

GOTO exit

END

if queue\_front == queue\_size - 1 THEN ; Front is at the rear of the circular queue, Loop front pointer of queue back to the front of the circular queue

queue\_front = 1

GOTO exit

ELSE

queue\_front = queue\_front + 1 ; Move front one back

GOTO exit

END

exit:

.END

.PROGRAM move\_kawa()

$movement\_state = "Initiating"

.current\_speed = 10 ;percentage of monitor speed

.current\_accuracy = 2 ;Milimeters

.current\_deceleration = 100 ;Percentage of max decel

.current\_acceleration = 100 ;Percentage of max accel

;

WHILE sig(keep\_active\_signal) DO

$movement\_state = "Waiting for trajectory point in queue"

IF queue\_front > 0 THEN

;

$movement\_state = "Setting motion parameters"

IF not queue[queue\_front, 1] == .current\_speed THEN

.current\_speed = queue[queue\_front, 1]

SPEED .current\_speed ALWAYS

END

IF not queue[queue\_front, 2] == .current\_accuracy THEN

.current\_accuracy = queue[queue\_front, 2]

ACCURACY .current\_accuracy ALWAYS

END

IF not queue[queue\_front, 3] == .current\_deceleration THEN

.current\_deceleration = queue[queue\_front, 3]

DECEL .current\_deceleration ALWAYS

END

IF not queue[queue\_front, 4] == .current\_acceleration THEN

.current\_acceleration = queue[queue\_front, 4]

ACCEL .current\_acceleration ALWAYS

END

;

IF queue[queue\_front, 0] == 11 THEN

POINT .#next\_joint\_position = #PPOINT(queue[queue\_front,6], queue[queue\_front,7], queue[queue\_front,8], queue[queue\_front,9], queue[queue\_front,10], queue[queue\_front,11])

JMOVE .#next\_joint\_position

END

IF queue[queue\_front, 0] == 12 THEN

POINT .next\_cartesian\_position = TRANS(queue[queue\_front,6], queue[queue\_front,7], queue[queue\_front,8], queue[queue\_front,9], queue[queue\_front,10], queue[queue\_front,11])

LMOVE .next\_cartesian\_position

END

;

IF queue[queue\_front, 5] == 1 THEN

BREAK

END

;

$movement\_state = "Deleting trajectory point from queue"

CALL del\_front\_queue

ELSE

TWAIT 0.1

END

END

.END

.PROGRAM shutdown\_kawa () ;

;$shutdown\_reason = "Something went wrong :( ERROR"

;PRINT $shutdown\_reason

;PRINT $recv\_tcp\_serv\_state

;PRINT $movement\_state

PRINT "Shutting down!"

signal -keep\_active\_signal

.END

.PROGRAM Comment\_\_\_ () ; Comments for IDE. Do not use.

; @@@ PROJECT @@@

; @@@ HISTORY @@@

; @@@ INSPECTION @@@

; @@@ PROGRAM @@@

; 0:init\_kawa

; 0:recv\_tcp\_serv

; 0:send\_pos\_serv

; 0:read\_tcp\_buffer

; 0:encode\_pose

; 0:decode\_traj\_pt

; 0:ins\_rear\_queue

; 0:del\_front\_queue

; 0:move\_kawa

; 0:shutdown\_kawa

; @@@ TRANS @@@

; @@@ JOINTS @@@

; @@@ REALS @@@

; @@@ STRINGS @@@

; @@@ INTEGER @@@

; @@@ SIGNALS @@@

; @@@ TOOLS @@@

; @@@ BASE @@@

; @@@ FRAME @@@

; @@@ BOOL @@@

.END