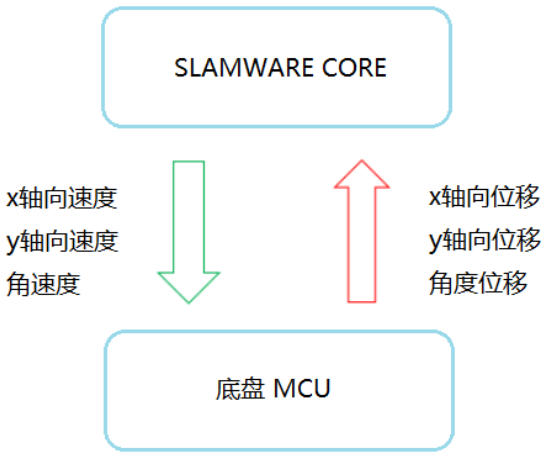
KBSW180145 Slamware

Breakout 6.0STM32SLAMWAREslamwarekit_reference_v6_code.20181120.zip

slamware1mm5%

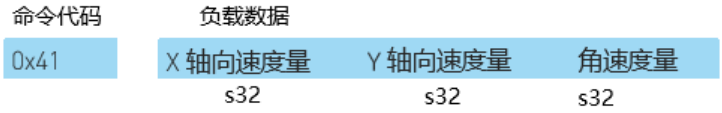
```
2/×0.001
```

> ***KBSW180148 Instruction for Integrating SLAMWARE Solution in Tri-omini-wheeled Base***



SLAMWARE Core deltaDeadreckon xyMCUDeadreckonxy(SET_V_AND_GET_DEADRECKON(0x41)

SLAMWARE CoreSET_V_AND_GET_DEADRECKON



X s32 X (m/s)Q16

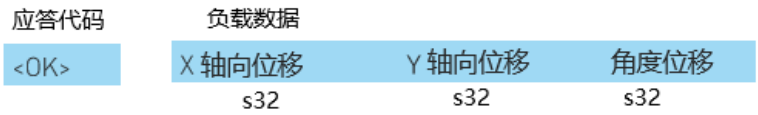
Y s32 Y (m/s)Q16

s32 (rad/s)Q16

0x41

```
typedef struct _base_set_velocity_request
{
    _s32 velocity_x_q16;
    _s32 velocity_y_q16;
    _s32 angular_velocity_q16;
} __attribute__((packed)) base_set_velocity_request_t;
```

SET_V_AND_GET_DEADRECKON, SLAMWARE Core



X s32 X (mm)Q16

Y s32 Y (mm)Q16

s32 ()Q16

0x41

```
typedef struct _base_deadreckon_response
{
   _s32 base_dx_mm_q16;
   _s32 base_dy_mm_q16;
   _s32 base_dtheta_degree_q16;
   _u8 status_bitmap;
} __attribute__((packed)) base_deadreckon_response_t;
```

SET_V_AND_GET_DEADRECKON

```
case SLAMWARECORECB_CMD_SET_V_AND_GET_DEADRECKON:
    {
            base_deadreckon_response_t ans_pkt;
            calculateMotorStatus(&ans_pkt);

             base_set_velocity_request_t *req = (base_set_velocity_request_t*)request->payload;
        float speed_l_mm = (float)req->velocity_x_q16 * 1000.0 / (1 << 16);
        float speed_r_mm = speed_l_mm;
        float line_speed_mm = (float)req->angular_velocity_q16 / (1 << 16) * robot_radius_mm;
        speed_l_mm -= line_speed_mm;
        speed_r_mm += line_speed_mm;

        set_walkingmotor_speed((int32_t)bumpermonitor_clamp_motorcmd(speed_l_mm), (int32_t)
bumpermonitor_clamp_motorcmd(speed_r_mm));
        if(speed_l_mm != 0)
        {
            last_motor_l_speed = (int)speed_l_mm;
        }
        if(speed_r_mm != 0)
        {
            last_motor_r_speed = (int)speed_r_mm;
        }
        ans_pkt.status_bitmap |= (is_ontheground()?0:BASE_MOTOR_TRACTION_LOST);  // 0ans_pkt.status_bitmap = 0;
        net_send_ans(channel, &ans_pkt, sizeof(base_deadreckon_response_t));
     }
     break;
```

1.dx, dy, dyaw

2.vx, vy, omegavl, vy

3.

1

变量名称	定义	单位
dl	左轮位移，机器人向前为正	m
dr	右轮位移，机器人向前为正	m
dx	机器人正方向位移，机器人前方为正	m
dy	机器人侧向位移，机器人左侧为正	m
dyaw	机器人角位移，逆时针为正	rad

2

dyaw=(dr-dl)/2R

dx=((dl+dr)/2)*cos(dyaw)

dy=((dl+dr)/2)*sin(dyaw)

R m

3

```
float d_yaw = (d_dist_r_mm_f - d_dist_l_mm_f)/2.0f/robot_radius_mm;
float displacement = (d_dist_l_mm_f + d_dist_r_mm_f)/2.0f;
float dx = cos(d_yaw)*displacement;
float dy = sin(d_yaw)*displacement;
ans_pkt->base_dx_mm_q16 = (_32)(dx*(1<<16));
ans_pkt->base_dy_mm_q16 = (_32)(dy*(1<<16));
ans_pkt->base_dtheta_degree_q16 = (_32)(d_yaw/M_PIF*180*(1<<16))
```

1

变量名称	定义	单位
vx	机器人正向线速度，前进方向为正	m/s
vy	机器人侧向线速度，向左为正, 对于二轮差动机器人，该值为 0	m/s
omega	机器人角速度，逆时针为正	rad/s
vl	机器人左轮线速度，机器人向前为正	m/s
vr	机器人右轮线速度，机器人向前为正	m/s

2 ()

vl = vx omega R

vr = vx + omega R

3

```
base_set_velocity_request_t *req = (base_set_velocity_request_t*)requestpayload;
float speed_l_mm = (float)req->velocity_x_q16 * 1000.0 / (1 << 16);
float speed_r_mm = speed_l_mm;
float line_speed_mm = (float)req->angular_velocity_q16 / (1 << 16) * robot_radius_mm;
speed_l_mm -= line_speed_mm;
speed_r_mm += line_speed_mm;
```

ODOMETER_EST_PULSE_PER_METER

```
=/(×)
```

Odometry

```
//
#define ODOMETER_EST_PULSE_PER_METER  6390UL


//60hz
#define CONF_MOTOR_HEARTBEAT_FREQ     60
#define CONF_MOTOR_HEARTBEAT_DURATION (1000/(CONF_MOTOR_HEARTBEAT_FREQ))


/*
 *
 */
static void _refresh_walkingmotor_odometer(_u32 durationMs)
```

```
{
    _u8 cnt;
    float dist_mm;    // disable interrupt
    _u32 irqSave = enter_critical_section();                                    //
    for (cnt = 0; cnt < WALKINGMOTOR_CNT; ++cnt)                          //
    {
        _lastEncoderTicksDelta[cnt] = _encoderTicksDelta[cnt];
        _encoderTicksDelta[cnt] = 0;
    }
    leave_critical_section(irqSave);

    if (durationMs == 0)                                                        //
        durationMs = 1;

    for (cnt = 0; cnt < WALKINGMOTOR_CNT; ++cnt)                          //
    {
        dist_mm = (float)_lastEncoderTicksDelta[cnt] * (1000.0/ODOMETER_EST_PULSE_PER_METER);
        _lastOdometerSpeedAbs[cnt] = dist_mm * 1000.0 / durationMs;
        dist_mm += _motorDistTailing[cnt];
        _motorDistAccumulated[cnt] += (_u32)dist_mm;
        _motorDistTailing[cnt] = dist_mm - (_u32)dist_mm;
        _motorDeltaTicks[cnt] += _lastEncoderTicksDelta[cnt];
    }
}

/*
 *
 */
float walkingmotor_delta_ldist_mm_f(void)
{
    _u32 delta_dist = _motorDeltaTicks[WALKINGMOTOR_LEFT_ID];  //
    _motorDeltaTicks[WALKINGMOTOR_LEFT_ID] = 0;
    return delta_dist * (1000.f / ODOMETER_EST_PULSE_PER_METER);
}


/*
 *
 */
float walkingmotor_delta_rdist_mm_f(void)
{
    _u32 delta_dist = _motorDeltaTicks[WALKINGMOTOR_RIGHT_ID]; ////
    _motorDeltaTicks[WALKINGMOTOR_RIGHT_ID] = 0;
    return delta_dist * (1000.f / ODOMETER_EST_PULSE_PER_METER);
}



#ifdef FEATURE_SET_V
static const float robot_radius_mm = 118.f;
static bool is_first_motor_data = true;
static int last_motor_l_speed = 0;
static int last_motor_r_speed = 0;


/*
 *
 */
static void calculateMotorStatus(base_deadreckon_response_t* ans_pkt)
{
    memset(ans_pkt, 0, sizeof(*ans_pkt));
    float d_dist_l_mm_f = walkingmotor_delta_ldist_mm_f();  //
    float d_dist_r_mm_f = walkingmotor_delta_rdist_mm_f();  //

    if(is_first_motor_data)
    {
       is_first_motor_data = false;
       ans_pkt->base_dx_mm_q16 = 0;
       ans_pkt->base_dy_mm_q16 = 0;
       ans_pkt->base_dtheta_degree_q16 = 0;
    }
```

```
    else
    {
        if(last_motor_l_speed < 0)
        {
            d_dist_l_mm_f = -d_dist_l_mm_f;
        }
        if(last_motor_r_speed < 0)
        {
            d_dist_r_mm_f = -d_dist_r_mm_f;
        }
        float d_yaw = (d_dist_r_mm_f - d_dist_l_mm_f) / 2.0f / robot_radius_mm;
        float displacement = (d_dist_l_mm_f + d_dist_r_mm_f) / 2.0f;

        float dx = cos(d_yaw)*displacement;
        float dy = sin(d_yaw)*displacement;

        ans_pkt->base_dx_mm_q16 = (_s32)(dx * (1<<16));
        ans_pkt->base_dy_mm_q16 = (_s32)(dy * (1<<16));
        ans_pkt->base_dtheta_degree_q16 = (_s32)(d_yaw / M_PIf * 180 * (1<<16));
           }
   }
#endif
```